AD-A118 731 PAR TECHNOLOGY CORP NEW HARTFORD NY F/G 9/2
ON-LINE PATTERN ANALYSIS AND RECOGNITION SYSTEM. OLPARS VI. PRO--ETC(U)
JUN 82 S E HAEHN, D MORRIS
UNCLASSIFIED PAR-82-15 NL

1 OF 3
AD A
118731

(2)

PAR TECHNOLOGY CORPORATION

Approved for public release:
distribution unlimited

2

OLPARS PROGRAMMER
AND
SYSTEM MAINTENANCE MANUAL

June 18, 1982

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD A115731	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) OLPARS VI (On-Line Pattern Analysis and Recognition System)		5. TYPE OF REPORT & PERIOD COVERED OLPARS Programmer and System Maintenance Manual
		6. PERFORMING ORG. REPORT NUMBER PAR 82-15
7. AUTHOR(s) Mr. Steven E. Haehn Ms. Donna Morris		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS PAR Technology Corporation Rt #5, Seneca Plaza New Hartford, New York 13413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of Defense Washington, D. C.		12. REPORT DATE June 18, 1982
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same As Block 11		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pattern Recognition, Structure Analysis, Discriminant Analysis, Data Transformation, Feature Extraction, Feature Evaluation Cluster Analysis, Classification Computer Software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The OLPARS Programmer and System Maintenance Manual (or Programmer's Reference Manual) contains the detailed information about the structure of all the files used in "portable" OLPARS. The manual also describes how to access the information contained in these files. A separate section describes the different "displays" OLPARS Generates, along with detailed information about the contents of the user's "display" files. Terminal and text-file input and output packages are described in detail here, also.

CLPARS Programmer and System Maintenance Manual

CLPARS VI

Submitted By

PAR TECHNOLOGY CORPORATION
Route 5, Seneca Plaza
New Hartford, New York 13413

Authors

Mr. Steven E. Haehn
Ms. Donna A. Morris

PAR REPORT #82-15

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

APPENDIX

FIGURE LOCATION

SECTION 1

INTRODUCTION

This document is meant to be used as a "reference" manual by CLPARS programmers or people who are to maintain an existing "portable" CLPARS system. Most of the overview information should be gotten from the CLPARS V Final Report.

This manual contains information only about the filing structure, I/O packages and RSX-11M specific features of "portable" CLPARS. Detailed information about specific CLPARS programs should be obtained from the CLPARS V and/or CLPARS VI Software Reference Manuals.

Within these pages you will find information about the structure of all the CLPARS files and how to access the content of these files. A separate section describes the different "displays" CLPARS generates, along with detailed information about the contents of a user's "display" files. Terminal and text-file input and output packages are discussed in detail here, also.

The basic ideas behind this current design of CLPARS are the maximum utilization of a standardized programming language (FORTRAN IV) for algorithm processing and the abstract treatment of the data by programs through the filing system and input/output processes. That is, all CLPARS processing routines will utilize subroutines for accessing data files and performing input/output.

Consequently, these processing routines can be machine and system independent and can be transferred to any computer configuration having a FORTRAN IV compiler. On each implementation, only a limited number of subroutines, which are machine or system dependent, will have to be reprogrammed. Thus, the key feature of this design is the concept of "portability".

1.0 DESIGN OVERVIEW

In this section, we present a brief overview of the design for the "portable" CLPARS; other sections will discuss individually, in greater depth, the key features of the design.

As mentioned in the Introduction, an important design ingredient in designing a machine and terminal independent, or "portable", CLPARS is the maximum utilization of FORTRAN as a standard, high level language which can be compiled on any machine on which CLPARS is likely to be implemented. This feature, the use of FORTRAN, is almost a design "must" in reaching the machine independent goal. However, since different computers have different operating systems (some computer installations may utilize more than one operating system with the same computer), and since operator terminals and other peripherals are not standard, independence within the filing system and I/O mechanism is difficult to achieve.

Therefore, in CLPARS, which is heavily file oriented and highly dependent on graphic I/O and operator interaction, the use of FORTRAN only partially meets the goal of portability. The other key aspects of the portable CLPARS design are: the separate program approach with a command input processor (CIP); the use of files for passing parameters and data; and an input/output package for performing I/O. Each of these areas will be discussed separately in subsequent sections.

1.1 THE COMMAND INPUT PROCESSOR (CIP)

From system to system, programs are initiated (started up) differently. The CIP is designed to overcome these differences. Each CLPARS function is designed as a separate program and should in itself be portable. In order to make the initiation of each program uniform, there is a system-dependent module, the CIP, which acts as an interface between CLPARS and the user. This mini-executive accepts the CLPARS program name and performs any system required operations needed to cause that program to run.

The CIP, along with the separate program approach, will support portability, modularity, ease of maintenance and expansion, and promote structural freedom in command execution. The CIP is discussed in detail in Section 2 and the design for the CIP under RSX-11M v3.2 can be found in Appendix A of this manual.

1.2 ABSTRACT FILES AND INPUT/OUTPUT

In order to accommodate various filing systems, word sizes and I/O peculiarities into the design of the portable CLPARS, each of the individual programs which implement a function will treat file data and I/O abstractly. That is, the programs will "know" what data is in a file, and will call a subroutine to store or retrieve the data. The subroutine, or possibly levels of subroutines, "know" the format of the files and how to access the physical data records.

I/O, particularly graphic I/O, will also be abstracted by subroutine calls. Thus, the individual programs which perform the CLPARS commands will be completely system independent; only the subroutines, which translate the abstract requests into specific code for a particular system, will be system or terminal dependent.

1.3 COMMAND STRUCTURE

The command structure (see CLPARS V Final Report) for portable CLPARS has been kept similar to that of the Multics CLPARS Operating System (MCOS). The complete structural freedom of MCOS has been retained. Complete designs for the programs that will carry out user commands and all necessary subroutines can be found in the CLPARS V-VI Program Specifications. Detailed descriptions of all the programs that have been written under the CLPARS VI contract can be found in the CLPARS VI Software Reference Manual (SRM). The CLPARS V Software Reference Manual contains program

specifications of all the programs initially designed for "portable" CLPARS. Note, the CLPARS-VI SRM is not a perfect subset of the CLPARS-V SRM.

1.4 CLPARS TRANSPORTATION CONSIDERATIONS

Ideally, the best way to transport CLPARS to a new operating system is to have two knowledgeable people; one who "knows" CLPARS inside out and one who "knows" the host operating system input/output (I/O) facilities inside out. As is usual when transporting application software, the I/O sections must be rewritten to accomodate the new operating system. Other system dependent considerations must also be taken into account. For instance, how are character strings represented in the local FORTRAN compiler? Is the length of the character string determined by counting the number of characters until an end-of-string symbol is encountered, or is there a portion of the string reserved to store the length attribute? Are the strings directly accesible to a program, or is there an indirect reference to the string through a string descriptor? Does the host operating system allow "spawning" or invocation of one program through another program (CLPARS may use spawning for a "clean" user interface)? Will the programs need to be "overlayed" and how easy will it be to overlay the programs? (nontrivial problem because reprogramming of entire command structure may be necessary).

In "portable" CLPARS we have isolated the I/O into packages. Section 6 of this manual explains in great detail what the results of the terminal and text file I/O package should look like. Section 3 explains the abstract concept of CLPARS "block" I/O package (FGET,FPUT) and Level I routines.

System dependent features of the portable CLPARS design are pointed out in many of the sections where they occur. A summary of the system dependent functions is given in Section 8. The appendices contain a discussion of the designs of how these system dependent aspects will operate on the PDP-11/70 under RSX-11M version 3.2. The CLPARS V Final Report (of November 15, 1979) gives an overview of the filing structure used with the details spelled out in sections 4 and 5 of this manual. An appendix in the CLPARS VI Software Reference Manual contains a list of names of all the system dependent programs.

SECTION 2

THE EXECUTIVE

2.0 INTRODUCTION

The goals of an executive for the portable FORTRAN version of CLPARS are to ease the user's access to the CLPARS routines, and to be simple enough to provide a maximum amount of portability. The latter goal is important since the executive and any accompanying overlay structure will be system dependent. In light of these goals, we have designed each user command to be a separate program and have written a small executive program to be an interface between the CLPARS user and the local "operating" system.

2.1 A COMMAND INPUT PROCESSOR (CIP)

To install CLPARS on most computer operating systems, a short executive called a Command Input Processor, or CIP for short, is written. The CIP, normally, is a system dependent executive program whose main jobs would be to accept user commands and parameters, add any system jargon (e.g., RUN, paths to an CLPARS directory), place any necessary parameters in a file, an then spawn (invoke or activate) the programs requested. For RSX-11M v3.2, the spawning capability has been left out of the CIP because it is handled very nicely by the system command file processor.

A COMMAND INPUT PROCESSOR (CIP)

By using a CIP, CLPARS "looks" the same on different operating systems. The user does not have to add RUN statements, directory information, etc., to his/her command line. Details about the Command Input Processor, under the RSX-11M version 3.2 operating system, can be found in Appendix A of this manual.

SECTION 3

FILE MANAGEMENT

3.0 INTRODUCTION

As we have seen in Section 2, CLPARS is considered as a set of individual programs, each of which performs a particular pattern recognition algorithm or an ancillary data manipulation function which supports the user and/or CLPARS as a system. Each of these programs corresponds to an CLPARS "command." The CLPARS user calls for the execution of these commands in a sequence which (s)he deems most appropriate for reaching a solution of a particular pattern recognition problem. In CLPARS, files of information play a major role in the system in several important ways: storing the basic input data; storing system information; storing decision logic and logic evaluation results; and storing intermediate information - results of one command which are used as inputs to subsequent commands.

As a result of using the approach described in this section, it will be possible for each program called by a user to be truly portable. All file handling will be accomplished by a limited number of subroutines, and only these will have to be rewritten when implementation is required on another machine/system. Furthermore, when writing the user-callable programs, it will not be necessary to know the details of the file formats - the

programmer need only know what information is stored in the files and utilize a set of machine-independent subroutines which access this information. Besides simplifying the portability problems, this, of course, makes implementation of new algorithms an easier task.

Additional benefits of this approach include minimizing physical record reads and writes, and an encoding scheme for naming files which eliminates the need for a program to do repetitious character string manipulation.

3.1 DEFINITIONS

The following definitions apply to any CLPARS user file.

The basic unit of a file is an "element." Each element in a file is sequentially numbered starting with 1 (one). An element can store a character, an integer, or a real number; the format of our element is the particular machine's floating point format. This definition eliminates problems of varying word sizes on different computers or the method of packing characters. Some space will be wasted by storing a single character as a floating point word, but since the great majority of information stored in CLPARS files is in the form of real numbers, the small inefficiency of storage utilization is more than off-set by the universality which results.

A file has a "header portion" which consists of 'n' elements (where 'n' could be zero), which contain information about the file as a whole - e.g., the number of entries in the file, or dimensionality of feature vectors contained in a file.

The remaining portion of the file (following the header) consists of "logical entries", or "entries" for short. Each logical entry contains 'k' elements ('k' is the same for each entry of a file; 'k', of course, can be different for different files). Entries are numbered sequentially starting at 1. It may be necessary to limit the maximum number of entries in a file to the largest positive integer of the machine's integer word format.

Two types of permanent files exist for an OLPARS user; these types are referred to as "fixed" and "variable." Fixed files are random access files that must exist in each user's directory and contain information on the status of that user's OLPARS files. There is a determined number of fixed files with fixed (or known) names. Some examples of fixed files are:

CM	Communications File
TL	Tree List File
LL	Logic List File
DI	Display Information File

Variable files are also random access. The number and names of variable files in existence for a user is variable (or unknown).

Examples of variable files might be:

TI + TREEA	Tree Information file for data TREEA
TV + TREEA	Tree Vector file for data TREEA
TI + TREEXYZ	Tree Information file for data TREEXYZ
TV + TREEXYZ	Tree Vector file for data TREEXYZ
LI + FIRSTTRY	Logic information file for a logic called FIRSTTRY

The plus symbol (+) denotes a sense of concatenation. This is necessary to allow for the many data trees and logics that a user may have in his/her directory.

Temporary random access files can be created by the CLPARS application programs for specific purposes. In addition, sequential files are used in the process of creating printer output of data.

3.2 FILE SYSTEM ROUTINES AND USAGE

There is a basic set of system dependent file manipulation and accessing routines within CLPARS. The manipulation routines are called COPEN, CCREAT(E), CCLOSE, CDELET(E), CRENAM(E), and CMOVE. These routines perform the following functions:

COPEN	Open an existing "system" file
CCREAT	Create and open a new "system" file
CCLOSE	Close an opened "system" file
CDELET	Delete or remove a "system" file
ORENAM	Give a new name to a "system" file
CMOVE	Move one "system" file to another (rename a "system" file to another existing "system" file)

The basic file accessing routines are called FGET and FPUT. These two routines control the actual movement of data between the computer memory and the CLPARS variable and fixed files.

All the above mentioned file manipulation and accessing routines (written in FORTRAN) are at the lowest level in the hierarchy of CLPARS file system routines. Therefore, these routines will be referred to as level I routines.

At a step above the system dependent file routines will be a set of routines that the CLPARS application programs use to invoke the more primitive level I routines. These programs will be known as level II routines. The level II routines have the quality of being "system independent." A few examples of some of these routines are as follows:

OPENTR	Opens an CLPARS data or logic tree (manipulation routine)
CREATR	Creates an CLPARS data or logic tree (manipulation routine)
OPENFX	Opens an CLPARS fixed file (manipulation routine)
TIGCAP	Gets the counts and pointers from a node entry in the tree information file (access routine)

Note that the level II file manipulation routines may use level II file accessing routines. OPENTR is still considered a level II file manipulation routine because it uses COPEN to perform the actual system file open. That is, OPENTR is a level II routine because it's a step above its corresponding file manipulation routine, COPEN, which is a level I routine.

The following paragraphs give additional characteristics of these routines.

- Some Level II file accessing routines may have been designed to perform format conversions when desired by the applications program. For example, if an applications program requests the number of nodes in a data tree to be returned into an integer variable N, then the level II program it calls will retrieve the number of nodes from the proper tree information file. Since all data is stored as "real," the level II program must convert the number of nodes to an "integer" and place it in N.

- Applications programs need not "know" the format of files; i.e., they do not need to know which element of a logical entry contains a particular item of data. They do need to know the name and calling sequence of the level II file access routine. The reason for this is that the level II file accessing routines "know" the format of the OLPARS files. The application program need only "know" the name and the calling sequence of the level II file accessing routine.

- o If more elements are added to a logical entry, previously designed level II file, accessing routines may not require modification (i.e., if the additions are made at the end of the logical entry, as opposed to being inserted in the middle of the logical entry).

- o All files utilized with this system will have the same length physical record. Level II file accessing routines are unaware of this length. Maximizing this length improves efficiency by minimizing actual file accesses; minimizing this length leaves more computer memory space for applications programs. Physical record length is a machine-dependent value. (See Appendix F for detailed CLPARS I/O notes on the RSX-11M operating system.)

Before giving a more detailed description of the file system routines, two tables, which are used by these routines, must be described.

The first table, called the File Code Table (FCT), is used to relate a number to a file name known to the particular operating system. Thus, the portable CLPARS system will be able to refer to all files in a simple, machine/system-independent numeric code

referred to as the file code (or FCD). The File Code Table, to which the FCD serves as the entry index, contains the type of information which is required for accessing real files for the "local" operating system.

All fixed files, which are described in later sections, have the following FCDs:

CM = 1	FV = 6
TL = 2	S1 = 7
LL = 3	SV = 8
DI = 4	SM = 9
DV = 5	HS = 10

For variable files, another scheme is used. For each user data tree there is an entry in the Tree List (TL) file, and for each user logic tree there is an entry in the Logic List (LL) file. These entries contain the tree name and FCDs for the two files, representing the data or logic tree. Note, the file codes for variable files are not predetermined like the file codes for fixed files.

The File Code Table is system dependent and accessed directly by the level I manipulation routines only. Appendix D gives the format of the FCT as it will appear under the RSX-11M operating system.

The second table used by the file routines is the File Access and Control Table (FACT). This table serves several purposes, all generally related to the set of files which are "open" or accessible to an OLPARS program during its execution. The entry index to this table is a File Descriptor number, or FID. An entry in FACT contains the "system" file name, the "logical unit number" assigned to the file (LUN), space for a physical record (PHBUFF), the record number of the physical record currently in the buffer (CPRN), the number of elements in the file header (HNE), the number of elements in a logical entry of the file (LENE), and a "put" flag (PF) which indicates that a change has been made by a "file put" function in some portion of the current physical record. (See Figure 3-1) The FACT table is incorporated into a common area that is available to all level I routines.

3.2.1 Level I Manipulation Routines -

COPEN locates the first empty entry in the File Access and Control Table (FACT) and sets a file descriptor (FID) to point to that entry. The file code (FCD) for variable files is obtained from the tree or logic list files before calling COPEN. The header size (HNE, header number of elements) and logical entry size (LENE, logical entry number of elements) of the file must be given so they

FIL							
1	LUN	filename	CPRN	HNE	LENE	PF	PRBUFF
2	LUN	filename	CPRN	HNE	LENE	PF	PRBUFF
.							
.							
.							
N (=15)	LUN	filename	CPRN	HNE	LENE	PF	PRBUFF

Figure 3-1 File Access and Control Table (FACT)

can be entered into the FACT. The "put flag" (PF) and the number of the current physical record (CPRN) entries of the FACT will be set to zero. CCPEN obtains a "system" file name from the File Code Table (FCT), via the FCD, and opens a "system" file. The logical unit number entry within the FACT is also set by CCPEN.

CCREAT(E) creates a new "system" file for CLPARS program use. CCREAT is given the name of the file to create along with the "type" of the file. The "type" indicates that the file is one of the following:

1. tree information file
2. tree vector file
3. logic information file
4. logic value file
5. fixed file

All the above types have entries placed in the File Code Table (FCT). The entry in the FCT is a "system" file name, created using the "NAME" and "TYPE" arguments of CCREAT. The file code of the created file is returned to the calling program. CCREAT fills in the FACT entries and returns a FACT pointer just like CCPEN.

CCLOSE first checks the "put" flag entry in the FACT (for the given FID) to decide whether it needs to write out the current buffered record. It then frees the portion of the FACT, pointed to by the given file descriptor, and performs a "system" file close.

CDELET removes an existing "system" file from the computer operating system's file structure. It also removes an entry in the File Code Table pointed to by the given file code argument, if the deleted file is a variable file.

CRENAM changes the name of the "system" file name within the File Code Table pointed to by a given file code.

CMOVE moves the "system" file, pointed to by one file code, to the "system" file, pointed to by another file code. This essentially is a "system" renaming function, but done in a different manner than CRENAM. (CRENAM is used when the new name of the file being renamed is not already in the FCT file. To insure that there are no duplicate file names, CRENAM must check the entire FCT file.) The file pointed to by the second file code is eliminated from the "system."

General Notes: The contents of a file cannot be accessed unless it is open at the time of access (the maximum number of files which can be opened simultaneously is limited by the number of entries in the FACT and any particular system limitations). A file should not be deleted if it is open. File deletion does not remove an entry from the TL or LL file; that is the application program's responsibility. In renaming files, the old file should be closed before it is renamed; the new file is not opened by CRENAM.

3.2.2 Level 1 File Access Routines -

One of the features of FGET and FPUT is the minimization of actual physical record accesses. A description of the FGET procedure follows.

FGET uses a file descriptor (FID) to determine which FACT entry describes the file to be accessed. From the procedural arguments specifying the logical entry of the file and the first element within that entry, plus information in the FACT entry (number of elements in header and number of elements in logical entry), it is possible for FGET to compute the "absolute" element address in the file of the first element to be retrieved. Dividing this address by the number of elements per physical record (constant for all files) will give the physical record number containing the first element. Comparing the required physical record number to the physical record number currently contained in the memory resident buffer determines if the element requested is already in memory; if it is, the element is returned to the calling program; if it is not, a new physical record must be read into the buffer. However, before reading the new physical record, a check of the "put flag" determines if any elements of the current physical record were changed by a write (FPUT) operation. If so, the memory buffer is written to the file before the new physical record is read into the buffer.

FPUT works in a similar manner. FPUT computes the "absolute" element address, the physical record number and the "relative" address in the physical record number of the first element to write. The computations are the same as those done in FGET.

If the required physical record number is not in the FACT physical record buffer (memory resident), the "put flag" is checked to see if it is necessary to write out the current physical record buffer before reading in the required record. If the "put flag" is set, the memory buffer is written to the file. The required physical record is then read into the memory buffer. If an end-of-file occurs, it is assumed that the file was opened for create, and the memory buffer is zeroed out. Any other type of read error causes FPUT to exit. If the required physical record number is memory resident, no read is necessary.

The elements to be written to the file are transferred from the calling routine buffer to the FACT physical record buffer. The "put flag" is set to indicate that elements have been changed. If there are more elements to be written (elements to be changed cross block boundaries), the current physical record buffer is written to the file and the next required record is read into the memory buffer. The process continues until all elements have been changed.

3.2.3 Level II Routines -

Since there has been a large number of level II routines developed to meet the needs of the CLPARS applications programmer, it is impossible to provide routine names and parameters in this document (see CLPARS V-VI Software Reference Manuals). However, several realistic examples are shown.

A routine naming convention is used, because of the number of routines possible. Those level II routines that read from or write into a file have names of five or six characters of the form FFXMM or FFXMMM, where FF represents the file name type, X is G for get (read) or P for put (write), and MM or MMM is a mnemonic for the routine.

For example, there can be a routine which returns, as an integer, the number of entries in the tree list (TL) file. The code for this routine is suggested below.

```
INTEGER FUNCTION TLGNOE (FID)
INTEGER FID, N
N = FGET (FID, C, 1, 1, X)
TLGNOE = X
RETURN
```

For this example, we have assumed that the number of the entries' value of the TL file is contained in the first element of the

header. The value of FID, of course, came from opening the tree list file. This example illustrates that a level II routine can change the real format used for all file elements.

As another example, we need a routine to retrieve the mean vector for a data set, given that we know the logical entry (LE) for this data set in the TI file.

```
SUBROUTINE TIGMN (FID, LE, NDIM, ABUF)
INTEGER FID, FGET
N = FGET (FID, LE, 17, NDIM, ABUF)
RETURN
```

In this example, we have assumed that the mean vector starts the 17th element of a TI file entry.

SECTION 4

PORTABLE CLPARS FILING STRUCTURE

4.0 INTRODUCTION

Each CLPARS user will have the following files in his/her user's directory:

Communications	(CM)
Tree Information	(TI)
Tree Vector	(TV)
Tree List	(TL)
Logic Information	(LI)
Logic Value	(LV)
Logic List	(LL)
Saved Vector	(SV)
Saved Transformation Matrix	(SM)
Display Information	(DI)
Display Value	(DV)
Projection Vector	(PV)
Scratch 1	(S1)
History	(HS)

The contents of the first nine files are described in detail in this section. The display files are discussed in Section 5. The history file is described in Appendix E, under the CLPARS

instrumentation package.

4.1 THE COMMUNICATIONS (CM) FILE

The CM file contains basic CLPAR system information that will be accessed by almost all of the CLPARS command programs. Since it contains no "data", it only has a header. This header is pictured in Figure 4-1. The following paragraphs explain the concepts of current data set, current logic and current major CLPARS option.

The current data set is the treename, nodename pair that the user works with. It consists of all the vectors and structure underneath that particular node. It is assigned using the SETDS command, which prompts for the name of a data set.

The current logic is the logic the user is in the process of designing on the current data set, or using to evaluate a test data set. In the portable FORTRAN version of CLPARS, we allow more than one logic file per data set. To do this, the user will be forced to assign unique names to all logic files. To begin a new logic the command NAMELOG is used; to restore an old logic the command SETLOG is used.

The user will be able to store several logics for the same data set. They may be complete or incomplete logics. If they are incomplete, they may be restored as the current logic and completed. When a logic is being designed or evaluated, its design set or test set must be the current data set.

	Element	
	1-8	Current Treename
	9	Tree Information File FCB
Current Data Set Info.	10	Tree Vector File FCB
	11-14	Current Node Name
	15	Entry Table Slot Number of Current Node
	16	NDIM
Current Logic Info.	17-24	Current Logic Name
	25	Logic Information File FCB
	26	Logic Value File FCB
	27	Number of Incomplete Logic Nodes
	28	Empty (area for future use)
	29-47	Screen Coordinate Information
	48	Prompt flag (0 = short prompt) (1 = long prompt)
	49	One-space Bin Factor (default = 5)
	50	Two-space Cutoff Value (default = 500)
	51	Instrumentation flag
	52	Instrumentation Threshold
	53	Current CLPARS Option # ...
	54-63	... and Option Name
	64-97	CLPARS directory path string

Figure 4-1 The Communications File Header

The current major CLPARS option is the last major CLPARS command that was used. It is "remembered" so that a correct set of subsequent options may be displayed to the user. The determination of the set of major CLPARS options and their option numbers can be found in the OPTION test file (see Appendix E).

For an explanation of the prompt flag in element 48 of the CM file, see Section 7.8.3 of the CLPARS V Final Report. The numbers stored in elements 49 and 50 relate to CLPARS displays. The use of the one-space bin factor is described in Section 5.2. The two space cutoff value determines whether a cluster plot or a scatter plot is produced on a two space display. If the number of vectors in the data set is less than the cutoff value, a scatter plot will be produced; otherwise, a cluster plot will be produced. The default cutoff value is 500. The instrumentation flag is used to enable and display the CLPARS instrumentation package. The prompt flag, bin factor, cutoff value, and instrumentation flag may be altered by using the command CDEFAULT.

4.2 THE TREE INFORMATION (TI) FILE(S)

The Tree Information file contains all necessary structural information about a data tree as well as the mean vector and covariance matrix for each node of the tree. The TI file consists of a fixed-length header portion and a fixed-length entry for each node in the tree (see Figure 4-2).

	Element	
	1	Number of nodes in tree
	2	Dimension (NDIM)
	3	Next available node position
	4	Next open entry at end of file
	5	Senior node pointer
	6	Entry Number or 0
	7	Entry Number or 0
	.	
	.	
	.	
	.	
Entry Table		
	Max+5	Entry Number or 0
	(Max = 100)	

Figure 4-2 The Tree Information File Header

The header contains the number of nodes in the tree including the senior node (in element 1), the dimension of the vectors (in element 2), the position in the TI file of the next available (empty) entry (in element 3), and the next open entry at the end of the file (in element 4).

In order to discuss the pointer in element 5 of the header, we must first explain the entry table and the meaning of any pointer in the TI file. When a node is placed in the TI file (initially, or by some tree manipulation routine), it is identified with a slot in the entry table. This slot contains the actual entry number that the node resides at within the TI file. Any pointer to a node will point to the slot in the entry table with which it is identified and not to the actual entry in the file. The slot with which a node is identified will not change, unless, of course, the node is deleted, in which case the corresponding slot will be zeroed.

The key feature of this scheme is that all pointers are independent of the actual entry position of the nodes in the TI file. Nodes can be physically moved within the TI file and pointers residing within the node do not have to be changed. The only change necessary is a change to the slot content in the entry table with which the nodes are identified. The addition of the entry table to the TI file header has greatly simplified many of the tree manipulation routines.

The senior node pointer in element 5 of the header then points to the slot in the entry table that is identified with the senior node. The senior node is accessed by first obtaining its entry position from that slot. The entry table follows. It contains slots for 100 nodes, which is the maximum number of nodes that a data tree may possess in portable CLPARS.

For each node in the tree, there will be an entry in the TI file (see Figure 4-3). The large set of pointers is designed to facilitate movement through the tree and, in particular, to enable programs to have easy access to the set of lowest nodes. The structure is completely general; any node of the tree can be considered as the senior node of a complete tree. A diagram of the pointers contained in elements 9 through 14 of the TI file entries is exhibited in Figure 4-4 (remember that pointers actually point to slots in the entry table).

The vector pointer in element 15 is the entry number of the first vector in the list of vectors (within the TV file) corresponding to a given lowest node. If the node is not a lowest node, this pointer is 0. For debugging and maintenance reasons, each node points (in element 16) back to the slot in the entry table with which it is identified.

	Element		Abbreviation	Further Description
	1-4	NODE NAME		
COUNTS	5	# of vectors at node	NOVECS	
	6	# of lowest nodes beneath this node	NLNODE	0 if lowest node
	7	# of children	NKIDS	0 if lowest node
	8	node level	NL	1=level of senior node
POINTERS	9	Parent Pointer	PP	0 if senior node
	10	Sibling Pointer	SP	points to next sibling; 0 if none
	11	First Child pointer	FCP	0 if lowest node
	12	First lowest node pointer	FLNP	0 if lowest node
	13	Lowest node link pointer	LNLP	0 if not a lowest node, -1 if last lowest node
	14	Lowest node back pointer	LNBP	0 if not a lowest node, -1 if first lowest node
	15	Vector Pointer in TV node	VP	0 if not a lowest node
	16	Back Pointer to Entry Table	BPET	Points to the slot in the entry table with which the node is identified
		MEANS		
	16+NDIM	COVARIANCES		
	END			

$$END = 16 + NDIM + \frac{NDIM(NDIM+1)}{2}$$

Figure 4-3 The Tree Information File Entry

```
                             HEADER
                               |
                               |
                  =============|==============
                 !!                          !!
                 !!      Senior   Node       !!
                 !!                          !!
                  ==========|====|===|=========
                   F   f    P    P   P
                F   f    P       P       P
              F   f    P         P           P
            F   f   F            P               P
          F   f  P               P                   P
        F  ===|=====         ==========           ==========
      F   !         !ssssss-!          !ssssssss-!          !
     F    !  Node   !       !   Node   !         !   Node   !
    F     !         !       !          !         !          !
   F       ==|===|==         =|====|===           ===|====|=
  F      F f F   P            l b  b l           Ff  P    P
  F    F f P     P            l b  b l         F f  F       P
  F  F f P       P            l b  b l        F f  F          P
  F F f F        P            l b  b l      F  f  F             P
 =|=|=|===      =======       l b  b l     F  =|=====        =======
!         !sss-!       !      l b  b l    F  !       !sssss-!       !
!   Node  !lll-! Node  !lllllll b  b l   F   ! Node  !      ! Node  !
!         !-bbb!       !-bbbbbbbb  b l   F   !       !      !       !
 =========      =======            b l   F    ====||=        ====|==
                                   b l   F    F f PP             l b
                                   b l   F   F f P   P           l b
                                   b l   F  F f P      P         l b
                                   b l   F F f P         P       l b
                                   b l  =|=|=|===       =======  l b
                                   b l !         !ssss-!       ! l b
                                   b l-!  Node   !llll-! Node  !ll b
                                   bbbb!         !-bbbb!       !-bbb
                                        =========       =======
```

P = parent pointer
s = sibling pointer
f = first child pointer
F = first lowest node pointer
l = lowest node link pointer
b = lowest node back pointer

(the - or ¦ symbol used at the end of a pointer means that the pointer is pointing to the node the - or ¦ is touching)

Figure 4-4 Pointers in the TI file entries

4.2.1 Additional Considerations -

a. The name of each node must be distinct and consist of one to four characters (blank padding will occur for "short" names). The first character of the name will be the one displayed on one-space or two-space displays. There are 46 displayable characters in the 1977 FORTRAN standard. Since only lowest nodes are used in such displays, a maximum of 45 lowest nodes will be allowed ('?' is used for prompting purposes). An intermediate node can begin with the same character as another intermediate or lowest node, but two lowest nodes must begin with distinct characters. The limitation on lowest nodes implies a maximum of approximately 90 nodes in an OLPARS data tree. These limitations can be eased (or further restricted) by changing the program (UNIQND).

b. Routines that perform manipulation with a data tree usually store the entry table within a program array, so as to have immediate access to the entry position of all nodes in the tree.

c. Before nodes are entered into a newly created TI file, the entry table is zeroed out so that no extraneous information exists in it.

d. When a node is deleted, or when several nodes are combined into one node, entry spaces may be left in the TI file. In these cases, it is up to the individual program to compress the file or to leave the spaces. In the latter case, element 3 of the TI file header will point to the first free entry and the rest of the free entry areas will be linked. The last free entry area will always occur after the last filled entry, and is pointed to by element 4 of the TI header. For a more complete discussion, see Section 4.3.

e. Storage of the Covariance Matrix:

Let C be the covariance matrix for a class of vectors of dimension n with elements C(i,j). Since C is symmetric (i.e., C(i,j) = C(j,i) for all 'i' and 'j' from 1 to 'n'), it is only necessary to store the lower triangular portion of C. We shall do this in a one-dimensional array, called A.

The order in which C is stored in A is C(1,1), ..., C(n,1); C(2,2), ..., C(n,2); C(3,3), ..., C(n,3); ..., C(n-1,n-1), C(n,n-1), C(n,n). That is, the columns of the lower triangular portion of C are stored one after the other in their natural order. The problem is to find a formula for a given C(i,j) that will yield a 'k' such that

$$C(i,j) = A(k)$$

The address of an element of the last row and jth column of an (n x n) matrix is given by

$$Q(n,j) = jn$$

To address a position 'i' in the jth column that is not in the last row (i < n), we have to subtract off the distance (n-i) which gives us

$$Q(i,j) = jn - (n-i)$$

Note, when 'i' equals 'n' we get our first equation again.

Now, when we deal with only the lower triangular portion of a square matrix, we will have to subtract off a little more area. Again, we start with the formula that gets us the address of the last row in the jth column of a square matrix and modify it so it looks as follows

$$C\ (i,j) = jn - \langle extra \rangle$$

where <extra> means all the array positions in the <u>upper</u> triangular region of the matrix, preceding (and including) the jth column (see Figure 4-5)

$$\langle extra \rangle = (j-1) + (j-2) + \ldots + 1$$

The sum of the numbers 1 to S is given by

$$\text{Sum}\ (S) = S\ (S+1)/2$$

Therefore, $\langle extra \rangle = (j-1)\ (j)/2$

and $C\ (n,j) = jn - j(j-1)/2$

Column #

Row #	1	2	3	4		n
1	1					
2	2	n+1				
3	3	n+2	2n			
4	4	n+3	2n+1	3n-2		
					Array Positions in 'A'	
n	n	2n-1	3n-3	4n-6		$\frac{n(n+1)}{2}$

Figure 4-5 The Positions of the Covariance Matrix (C(ij), i>=j), in the array A

To address position 'i' in the jth column of the lower triangular portion of this matrix ($i >= j$) that is not in the last row ($i < n$), we have to subtract off the same distance ($n-i$) as we did in the previous case. Hence, for ($i >= j$), the position of C (i,j) in array A is

$$k = jn - j(j-1)/2 - (n-i)$$

4.2.2 Numerical Example -

Consider the following simple data tree called XEXAMPLE of dimension four.

```
                          o  **** (senior node)
                 ---------|---------
                /         |         \
               |          |          |
               |          |          |
               o  ANCD    o  BNCD    o  CNCD
          -----|-----
         /           \
        |             |
        |             |
        o  DNCD       o  ENCD
```

DNCD and ENCD have been created by structure analysis performed on ANCD. The contents of the TI file for this example are shown in Figure 4-5a.

Element	Header	**** ENTRY 1	ANCD ENTRY 2	BNCD ENTRY 3	CNCD ENTRY 4	DNCD ENTRY 5	ENCD ENTRY 6	
1	6	*	A	B	C	D	E	N
2	4	*	N	N	N	N	N	A
3	7	*	C	C	C	C	C	M
4	7	*	D	D	D	D	D	E
5	1	225	75	75	75	35	40	Number of Vectors
6	1(*)	4	2	0	0	0	0	Number of Lowest Nodes
7	2(A)	3	2	0	0	0	0	Number of Children
8	3(B)	0	1	1	1	2	2	NL
9	4(C)	0	1	1	1	2	2	PP
10	5(D)	0	3	4	0	6	0	SP
11	6(E)	2	5	0	0	0	0	FCP
12	0	5	5	0	0	0	0	FLNP
13	0	0	0	4	-1	6	3	LNLP
14	0	0	0	6	3	-1	5	LNBP
15	0	0	0	1	76	151	186	VP
16	0	1	2	3	4	5	6	BPET
17-20	0	MEANS	MEANS	MEANS	MEANS	MEANS	MEANS	
21	0	COVS.	COVS.	COVS.	COVS.	COVS.	COVS.	
30								
.	.							
.	.							
.	0							
MAX+5								

Figure 4-5a The TI File for XEXAMPLE

4.2.3 Creating New Trees From Old Trees -

The CLPARS data tree entry table plays an important role when creating a new data tree from a portion of an existing data tree. Its usefulness comes in the form of minimal structural pointer changes.

For example, the portion of an existing tree (TREE1) used to create a new tree (TREE2) contains nodes 2-4-5, where 2 is defined to be the "current data node" (see Figure 4-6). The slot numbers of the nodes that make up the new tree will be identical to their corresponding slot numbers found in the old tree. However, note that the contents of the entry table slots (entry numbers) in the new tree differ from those found in the old tree. The new slot contents relect the actual position of the nodes obtained from the old tree.

A few more changes are necessary to complete the transformation of old to new. Each node in the new tree requires a new node level. For those nodes that are lowest nodes, a new vector pointer is needed (i.e. vectors are transferred, too). The first lowest node in the new tree must have its lowest node back pointer set to -1 (end link indicator) and the last lowest node must have its lowest node link pointer also set to -1. The "current node" of the old tree is now the senior node of the new tree. Thus the node's name must be changed to '****', the name of the senior node in every CLPARS data tree. Also, the parent pointer sibling pointer of the new senior node must be set to zero,

```
                TREE1

                  1              portion of
                 / \            TREE1 used         TREE2
                /   \
               2     3    --->       2       --->    1
              / \                   / \             / \
             /   \                 /   \           /   \
            4     5               4     5         2     3

       (NOTE: numbers in the above trees are slot numbers)

                ---                                ---
               | 1 | senior slot                  | 2 | senior slot
               |---| -----------                  |---| -----------
slot nos.                            slot nos.
                ---     TREE1                      ---     TREE2
     1         | 1 | entry table          1       | 0 | entry table
               |---|                              |---|
     2         | 2 |                      2       | 1 |
               |---|                              |---|
     3         | 3 |                      3       | 0 |
               |---|                              |---|
     4         | 4 |                      4       | 2 |
               |---|                              |---|
     5         | 5 |                      5       | 3 |
               |---|                              |---|
     6         | 0 |                      6       | 0 |
                 .                                  .
                 .                                  .
                 .                                  .
```

Figure 4-6 Creating a tree relying on
            data tree entry table

indicating there are no parents or siblings. Finally, the senior node slot number (found in the TI header of the new tree) is set to point to the senior node.

By using this method of data tree creation, all the structural pointers in the new tree do not have to be changed. An example of this method can be found in the subprogram MXFRM.

4.3 CLPARS FILE "FREE" LISTS

In the logic and data tree files of CLPARS (and some "fixed" files), we have the concept of a "free" list of entries or nodes. These free entries or nodes at one time were "active" within the tree structure, but were subsequently deleted, or made "inactive". These inactive entries can now be used when a new entry or node is to be added to the tree.

Within the header of the tree files that have "free" lists (the Tree Vector file does not have a "free" list), resides the information necessary to maintain the list. This information consists of a pointer to the head of the free list, a pointer to the end of the free list (actually the end of the file), and the number of active entries or nodes in the tree.

Each entry or node in the "free" list has a pointer to the next entry or node in the list (this is a one way directional list). This pointer resides as the first element in the entry or node. (See Figure 4-7).

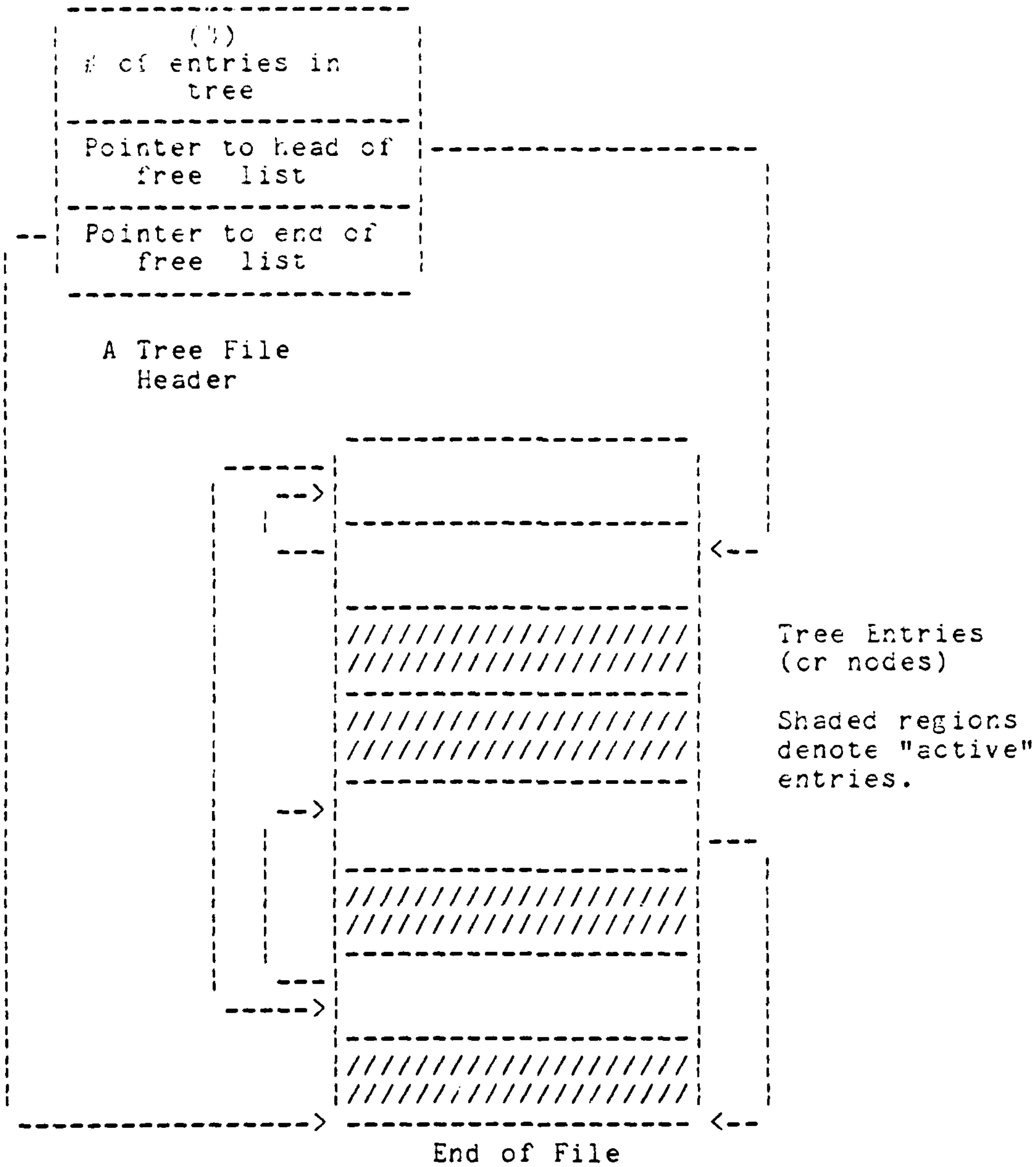


Figure 4-7 Free List of a Tree File

When the free list is empty, the pointer to the head of the free list and the pointer to the end of the free list will both be pointing to the end of the file When an "active" entry becomes "inactive" or deleted, it is entered into the free list in the following manner:

1. The pointer to the head of the free list is placed in the first element of the deleted entry.

2. The pointer to the head of the free list is changed so it points to the deleted entry.

3. The number of "active" elements in the tree is decremented by one.

When a new entry is to be added to a tree, the free list is consulted first.

If the pointer to the head of the free list is equal to the pointer at the end of the free list, then there are no "inactive" nodes within the tree. Therefore, the tree (file) is extended to accommodate the new entry, and both the "head" and "end" pointers of the free list are reset to point to the end of the file. The number of elements in the tree is incremented by one.

However, if the free list contains some "inactive" members (i.e., "head" not equal to "end"), the "head" pointer is used to obtain its new value from the "inactive" entry to which it is pointing. Again, the number of elements in the tree is incremented by one. (Note: the number of elements in a free list can be found by adding one to the total number of "active" elements in the tree and subtracting that quantity from the "end" pointer.)

4.4 THE TREE VECTOR (TV) FILE

The vectors of a data tree are stored in its Tree Vector file. The vectors are grouped together by lowest node to facilitate access to the set of vectors in a lowest node and to free the TV file of pointer storage (see Figure 4-8).

The header of the TV file contains the last logic designed (or evaluated) on the tree as well as the file codes of the files that make up that logic tree. It also contains a pointer to the end of the file. An entry in the TV file is identified with a vector. In it, the vector is stored along with its ID, the first character of the lowest node that the vector resides in and its temporary logic indicator (see Figure 4-9). The temporary logic indicator will contain the node number that the vector was placed at from the last logic applied to the tree (except for nearest neighbor logic). This element will be used during logic design or evaluation to keep track of the logic node at which the vector currently resides. This is necessary because a class may have vectors in several logic nodes during logic design, and during evaluation it is a convenient

Element		
1 thru 8	Latest Logic Name	Header
9	The File Code of the LI File	
10	The File Code of the LV File	
11	Next Open Entry at End of File	
	Vectors in a Node	
	Vectors in a Node	
	. . . . . .	

Figure 4-8 The Tree Vector File

1 rel.	1 rel.	1 rel.	(NDIM) rels.
Class Symbol	Vector Identifier	Logic Indicator	Vector Measurements

Figure 4-9 A single CLPARS data vector (as found in TV file)

way of keeping track of the results. Moreover, if a logic design is not completed in a given user session, then the lowest logic node that each vector resides in is saved in the temporary element until the next session. The logic will not have to be reevaluated before the logic design is resumed.

Note that the length of a TV file logical entry is three more than the dimensionality of the vector and essentially depends on the dimensionality. This dependency presents no difficulty since the dimensionality is stored in the Tree List file entry corresponding to the data set.

4.5 THE TREE LIST (TL) FILE

The Tree List file is a list of the data trees that the user has in his directory. It has a two element header that contains the number of entries (data trees) in the file, and an alphabetic list pointer. Each entry corresponds to a data tree. An entry contains two File Codes (FCD) representing the files that make up the data tree, the dimension of the tree, space for an 8-character name of the tree, and an alphabetic list pointer. See Figure 4-10.

When a data tree is added to the system, its Tree List entry is placed at the bottom of the Tree List file. When a data tree is deleted from the system, its entry in the Tree List file is filled by the last entry in the file, and the count is decremented by 1. Consequently, no free list is maintained in this file.

Element		
1	Number of Entries	Header
2	Alphabetic Link List Ptr.	
3	File Code (FCD) for TI File	Entry 1
4	File Code (FCD) for TV File	
5	NDIM	
6	Alphabetic Link List Ptr.	
7	T	
8	R	
9	E	
10	E	
11	N	
12	A	
13	M	
14	E	
. . .		Entry 2

Figure 4-10 The Tree List File

The alphabetic list pointers are used to keep the list entries in an "alphabetic" order. All new entries to the TL file are "placed" in the proper order via these pointers.

4.6 THE LOGIC INFORMATION (LI) FILE

For each logic tree that exists in portable CLPARS (i.e., for each logic tree listed in the Logic List (LL) file) there will exist a Logic Information (LI) file and Logic Value (LV) file. These files have a parallel in the TI and TV files associated with each data tree. The LI file contains general information about the logic tree and its structure; the LV file contains the actual logic (decision criteria) for each node of the logic tree. In this section the LI file is discussed.

The header part of the LI file has a fixed length; the length is a function of the maximum number of "displayable" classes (i.e., lowest nodes in a data tree) that are allowed for any data tree. Figure 4-11 shows the content of the header, assuming 50 is the maximum number of classes allowed.

Most of the content of the header is self explanatory. NAN (next available node, element 15) is the entry number (in the body of this file) for the next node to be defined. In general, NAN will be one greater than NNU (number of nodes used, element 17); it will only be different in the cases where certain nodes, already having logic defined, have been deleted. The current logic node in element 18 is only used in conjunction with the design of 1 or 2

Element		
1-12	Name of Design Data Set (DDS)	
13	NDIM - Dimensionality (of DDS)	Counts and Pointers
14	NCLAS - Number of Lowest Nodes (in DDS)	
15	NAN - Next Available Node Entry	
16	NOE - Next Open Entry at end of file	
17	NNU - Number of Nodes Used in Logic Tree	
18	Current Logic Node (Group 1-S and 2-S)	
19	RNF - Use Reassociate Names Flag (for Evaluation)	
20	LVEL - Number of Elements in LV File Entry	
21	NILN - Number of Incomplete Logic Nodes	
22	Blank	
23-26	Space for Name of First Class	DDS CLASS NAMES
.	(Only NCLAS Spaces Used for a Tree)	
219-222	Space for Name of 50th Class	
223	A Priori Probability - First Class	
.	(Only NCLAS Spaces Used for a Tree)	
272	A Priori Probability - 50th Class	

Figure 4-11 The Logic Information File Header

space group logic (this logic design is a two step process which might not be completed).

The entries in the body of the LI file each describe a node of the logic tree; the length of these entries is fixed and independent of any data set parameters (e.g., the vector dimensionality or the number of classes), but it is machine dependent because of the classes-present bit map, which is described later. The logic node number of a node will be the same as its logical entry number in the LI file. This gives a simple and unique way of assigning logic node numbers. Figure 4-12 shows the content and format of an LI file entry.

The following list shows the logic type code of the current OLPARS logics, along with the appropriate subtype codes. (Logic type and subtype codes are elements of the LI file entry).

Logic Code Type

- 0 - undefined (incomplete)
- 1 - pairwise logic
- 2 - group logic
    - 1 - one-space
    - 2 - two-space
    - 3 - Boolean
- 3 - nearest mean vector logic
- 4 - closed decision boundary
- 5 - nearest neighbor

Element	
1	Logic Type Code (or Entry Link)
2	Logic Sub-Type Code
3	Option Number of the routine that created the logic at this node (0 if lowest node)
4	Node Level (Senior Node is Level 0)
5	Number of Nodes Below this Node at Next Level
6	Entry Number of Parent Node (0 for Senior Node)
7	Entry Number of First Child Node
8	Entry Number of Next Sibling (0 if No Next Sibling Node)
9	Entry Number (in LV File) for Start of Decision Logic for this Node
10	Entry Number (in LV File) for Reject Strategy for this Node - 0 if No Reject Strategy
11-14	Original Design Data Set Class Name for this Node (only for lowest nodes)
15-18	Reassociated Class Name for this Node (only for lowest nodes)
19	Modified Logic Flag
20	Number of Classes Present at Node (0-for reject)
21- ?	Bit Map Designating Classes Present at this Node

Figure 4-12 The Logic Information File Entry

Since pairwise logic, nearest mean vector logic, closed decision boundary, and nearest neighbor logic have no subtypes of logic, their subtype code will be zero (means undefined).

Closed decision boundary, however, does have a "sub-block" code that is found in an element in the LV file (because nodes in a closed decision boundary logic are allowed to have more than one type of decision boundary). These sub-block codes are as follows:

1 - hyperrectangular sub-block

2 - hypersphere sub-block

3 - hyperellipsoid sub-block

If the value of the logic type code is negative, it indicates that the node had been defined and subsequently deleted and the value is now a link pointer to the next available node entry. This is used in conjunction with NAN and NNU elements of the header and prevents lost "holes" in the entry portion of the file.

Elements containing node level, first child node, and next sibling node facilitate searching the logic file entries for drawing, and listing or deleting logic nodes.

Elements 9 and 10 are the entry numbers in the LV file for the first entry containing the logic decision criteria and the logic for the reject strategy, if it exists for this node.

For completed logic nodes (i.e., nodes with only a single class present), elements 11 through 14 will contain the name of the class at this node (design data act class name). If it happens to be a reject node, the name of the "class" will be "****"; although this is also the name of the senior node, there is no conflict. The type of logic code for a completed node will be 0 or undefined, since there is no logic; there may be, however, a reject strategy defined. Elements 15-18 contain a class name to be associated with the original data class (reassociated class name). This name may be used during logic evaluation to associate a test data class to the original design data class. During logic creation the reassociated class name and the original design data set class name are identical. The reassociated class name may be changed by the command REASNAME.

The classes present bit map, which starts at element 21, is one place where it is necessary to deviate from the standard method of using real format words for each element of a file. This item is a binary string equal in length to the maximum number of classes allowed in the system. If the ith bit of the string is is a one, it indicates that the ith class (in the list of class names in the header, starting at header element 23) is present; if the bit is zero the class is not present. To use a single real word, for each class at each node, is felt to be excessive in storage space. Thus, a machine dependent (because of word length) scheme is utilized for this bit map. (On the PDP-11 machines this bit map will be contained in two real elements). For a reject node, the

bitmap is undefined and the number of classes present is zero. For a lowest (completed) node, the bitmap is redefined and the number of classes present is one. For this type of node, the first element of the bitmap contains a pointer into the list of design data set class names found in the LI header. The pointer can be used to obtain the name of the class that is associated with the logic node. The pointer is also used as an "assigned class" index during logic evaluation.

(LI File Initialization)

When a logic file is created by the command NAMELOG, the LI file and the LV file will be created. NAMELOG will set the following values in the LI file:

Name of design data set (DDS)

Data Set Dimensionality (NDIM)

The number of classes in the data set (NCLAS)

The next available node (NAN=2)

Next open entry at end of file (NOE=2)

Number of nodes used (NNU=1)

Reassociated Names Flag (RNF=0)

List of Lowest Node Class Names

List of Class A Priori Probabilities

The above values are placed in the header; those mentioned below are put in the first entry, which is the senior node of the logic tree.

Logic type code (= 0)

Node level (= 0)

Number of nodes below (= 0)

Entry number of parent (= 0)

Number of classes present (= NCLAS)

The bit map which indicates all classes present

4.7 THE LOGIC VALUE (LV) FILE

The LV file of a logic file pair contains the parameters or variables which define the actual decision logic criteria for any node of a logic tree. The entries are pointed to by the node entries in the LI file. Entries of the LV file also contain the blocks which specify reject strategy logic, if used at any node.

In keeping with the filing system conventions established for portable CLPARS, all entries in the LV file are of the same length; this length is established at the time of creation and is a function of the dimensionality of the data set on which the logic is being designed. However, the different types of logics require different amounts of storage. To accommodate this flexibility within the LV file, the design is such that a logic block (the definition criteria for the logic of a node) may be contained in a number of entries of the LV file.

The entry in the LV file for a node contains the entry number of the first entry of the logic block in the LV file. The first element of an LV file entry is a link which points to succeeding entries of that logic block. A zero in the first element indicates that this entry is the last entry of a logic block. This linking is necessary for the efficient deletion or change of the decision logic at a node.

The length of an entry in the LV file is NDIM+2 with a minimum length of 12 elements. This length was chosen after analyzing the space requirements for the different types of logic blocks and optimizing the choice of length as a trade off between unused space and complicated structure. The minimum length of 12 is needed in the Fisher pairwise logic block.

Entries that comprise a logic block do not have to be contiguous; since they are forward linked to each other, they do not have to be sequential. They will be non-sequential only in the cases where a previously designed logic has been deleted. Elements in the header and the link element of each entry implement this capability.

The LV file header contains three elements; these are defined in order below:

Element	Contents		Initial Value
1	NEU	total number of entries actively used	0
2	NOE	next open entry at end of file	1
3	NETU	next entry to use	1

NOE will differ from NETU only if there has been a deletion of defined logic. The total number of entries that could be found in a "free" list (deleted entries list) is found by NOE - (NEU + 1). Initial values, set when the LV file is created, are shown at the right.

The use of the first element of each entry, the next entry link or link element is similar in all cases. If the number is positive, it is the entry number of the next entry in the block; if it is zero, it is the last entry used by a block; if it is negative, it indicates that it was a deleted entry, but its absolute value is still treated as a link (now a link of deleted, and available for reuse, entries). The last entry of the string of deleted entries contains a link to NOE, the next entry after all used entries. (See Section 4.3 on "free" lists).

In addition to the link element which forward links all entries containing logic for a node, certain entries may contain a link to a particular portion of the logic. This is done to facilitate getting to certain sections of the more complicated types of logic or logics which permit suboptions (e.g., the NMV command). Figures 4-13 and 4-13a illustrate this multiple linking. Note, in the Fisher Pair logic block example in Figure 4-13A, there are three forward links with a value of zero; one for the main logic block, two for auxillary logic regions. Each represents the last entry of their respective regions.

Figures 4-14 through 4-24 describe the file entry formats for different types of logic blocks.

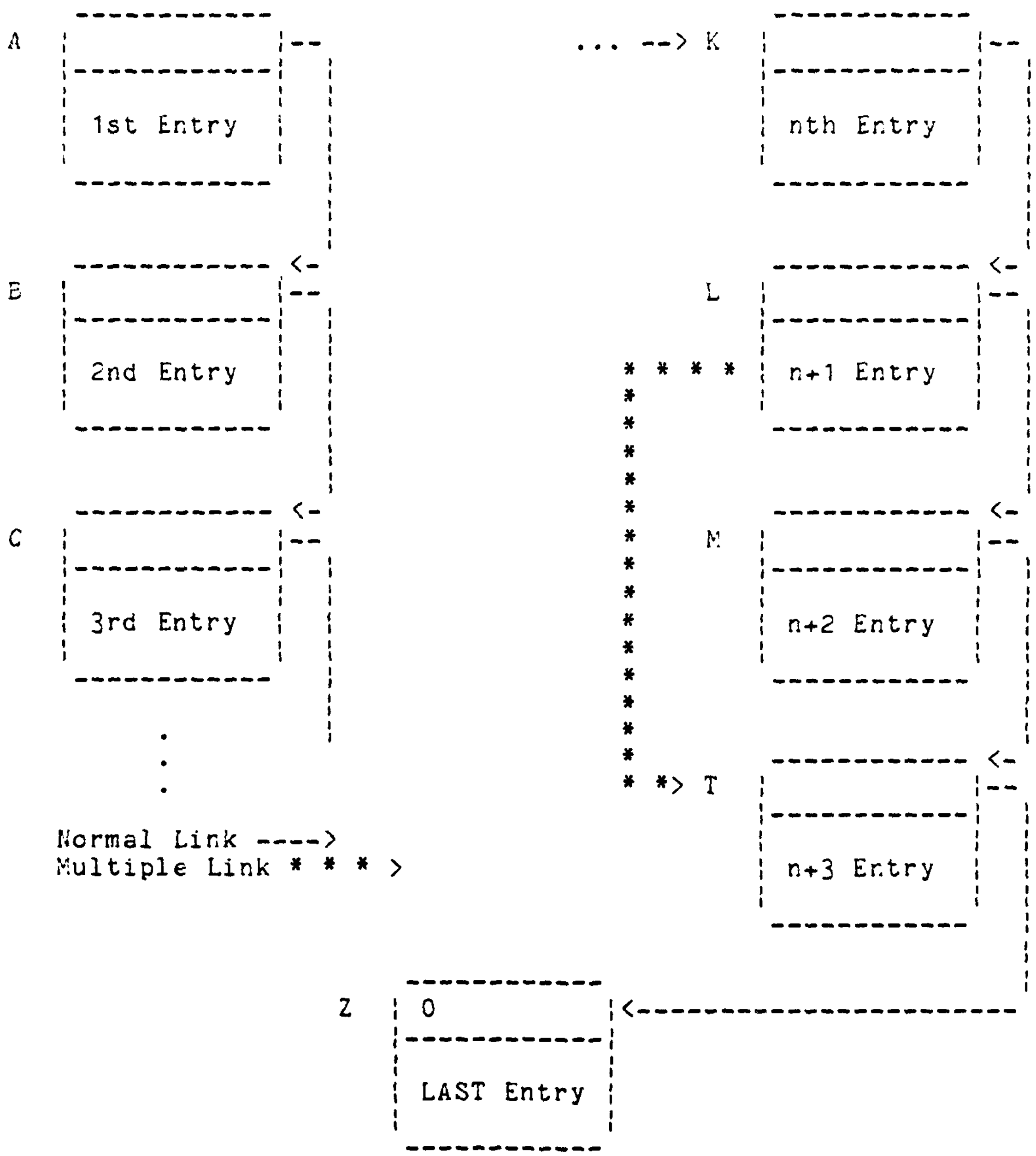


Figure 4-13 Multiple Linking Within A Logic Block

```
     Logic block
        header (pointer to here found in LI entry)
          |
     ///////////////---->/////////////////----> ...   //////////////---
     ///////////////     /                /           /            /  |
     ///////////////     / node number    /           / node nbr.  /  |
     / class pair  /     / associations   /           / assoc.     /  |
   --/ links ptr.  /     /                /           /            /  |
   | ///////////////     /////////////////            //////////////  |
   |                                                                  |
   -------------->//////////////////////////////////<-----------------
             ------/                                /--->... to next entry
             |  ---/        Class Pair Links        /        of class pair
from last    |  |  /                                /        links (CPL)
entry of     |  |  /                                /
CPL          |  |  //////////////////////////////////-------------
 |           |  |                                                 |
 |           |  |                                                 |
 |           |   --------------------                             |
 |           |                      |                             |
 |           V                      V                             V
  -->/////////////--    ----->/////////////--   --> ...  //////////////--
 ----/ aux. ptr. /  |  |      / class pair/  |  |    -----/ aux. ptr. /  |
|    /           /  |  |      /  header   /  |  |   |     /           /  |
|    /   (cph)   /  |  |      /   (cph)   /  |  |   |     /   (cph)   /  |
|    /////////////  |  |      /////////////  |  |   |     /////////////  |
|                   |  |                     |  |   |                    |
|  --/////////////<-   |   --/////////////<-    |   |   --/////////////<-
|  | /           /     |   | /           /      |   |   | /           /
|  | / fisher    /     |   | / fisher    /      |   |   | / fisher    /
|  | / direction /     |   | / direction /      |   |   | / direction /
|  | /           /     |   | /           /      |   |   | /           /
|  | /////////////     |   | /////////////      |   |   | /////////////
|  |                   |   |                    |   |   |
|  ->/////////////----     ->/////////////----      |   ->/// zero ////
|    /           /           /           /          |     /           /
|    / fisher    /           / fisher    /          |     / fisher    /
|    / orthoganal/           / orthoganal/          |     / orthoganal/
|    /           /           /           /          |     /           /
|    /////////////           /////////////          |     /////////////
|
|     c1 .vs. c2            c1 .vs. c3              |    c(n-1) .vs. c(n)
|                                                   |
 --->////////////--> ... /// zero ///                -->////// zero /////
     /          /        /          /                  /                /
     /  optimal discriminant logic  /                  / any auxillary  /
     /          /        /          /                  / logic region   /
     ////////////        ////////////                  //////////////////
```

Figure 4-13A Example of Fisher Pair Logic Block

Entry Number 1

Element	Content
1	Link Element - Entry Number of Second Entry
2	Number of Boundaries (= 1 or 2)
3	Node Number Associated with Right-Most Region
4	Node Number Associated with Left-Most Region
5	Node Number Associated With Middle Region (if 2 Boundaries)
6	Right Threshold Value
7	Left Threshold Value (if 2 Boundaries)

Entry Number 2

Element	Content
1	Link Element (= 0)
2 . . . NDIM+1	NDIM Discriminant (Projection Vector) Coefficients
NDIM+2	Number of Mesurements Used

Figure 4-14 One-Space Group Logic Block Format

Portable CLPARS Filing Structure -- 4
THE LOGIC VALUE (LV) FILE

Entry Number 1

Element	Content
1	Link Element
2	Number of Boundaries - NB (=1 to 2)
3	Number of Segments in First Boundary - NS(1) (= 1 to 5)
4	Number of Segments in Second Boundary - NS(2) (= 1 to 5) (= 0 if NB = 1)
5	Link Word to Entry Containing discriminant vector of First Segment of Second Boundary (if it Exists)
6	Link to Entry Containing First (and second, if it exists) boundary segment discriminant value thresholds.
7	Node Number Associated With Excess Region
8	Node Number Associated With Convex Side of Boundary 1
9	Node Number Associated With Convex Side of Boundary 2 (Ignored if NB = 1)

Remaining NS(1) + NS(2) Entries

Element	Content
1	Link Element
2 . . . NDIM+1	NDIM Discriminant Coefficients

Figure 4-15 Two-Space Group Logic Block Format

Last Entry

Element	Content
1	Link Element
2 - ... 10	Threshold Values (NOTE: 2nd-boundary thresholds appear immediately after those of 1st boundary, with no intervening space)

Figure 4-15 Two-Space Group Logic Block Format (continued)

Entry Number 1

Element	Content
1	Link Element
2	Number of Characters in Boolean Statements - NC (max 132)
3	Node Number Associated With True "Side"
4	Node Number Associated With False "Side"

Remaining Entries - Number Sufficient to Contain NC Elements

Element	Content
1	Link Element
2 - . . . NDIM+2	Characters of Boolean Statement

Figure 4-16 Boolean Group Logic Block Format

Entry Number 1

Element	Content
1	Link Element
2	Number of Classes - NCLAS
3	Weight Flag: 1 - No Weights, 2 - Weighting Vectors, 3 - Weighting Matrices, 4 - Quadratic Classifier
4	Link Element to First Entry Containing Means
5	Link Element to First Entry Containing Weighting Vectors
6	Link Element to First Entry Containing Weighting Matrices
7	Ignore Measurement Flag: 0 - Use All, 1 - Ignore Some
8	Reject Flag: 0 - No Reject Boundaries, 1 - Use Reject Boundaries
9	Link Element to First Entry Containing Reject Values
10	Link Element to First Entry Containing Determinants

Entry Number 2

Element	Content
1	Link Element
2 . . . NDIM+1	Measurements to be Ignored Flags = 1 Use Corresponding Measurement = 0 Ignore Corresponding Measurement
NDIM+2	Number of Measurements Used

Figure 4-17 Nearest Mean Vector Logic Block Format

Next Group of Entries - Number sufficient to contain NCLAS+1 Elements

Element	Content
1	Link Element
2	Node Number Associated w/rejects
3	Node Number Associated w/first class
.	
.	
.	Node Number Associated w/last class
.	

Next Group of Entries - Number sufficient to contain NCLAS elements

Element	Content
1	Link Element
2	
.	
.	NCLAS reject boundary distance values (square of value entered by user)
.	

Next NCLAS Entries - One entry per class

Element	Content
1	Link Element
2	
.	
.	NDIM Mean Components
.	
NDIM+1	

Figure 4-17 Nearest Mean Vector Logic Block Format (continued)

Next NCLAS Entries - One entry per class

```
Element        Content
-------        -------
   1           Link Element

   2   -
       |
   .   |
   .    >      NDIM Weighting Components (variance vector)
   .   |
       |
NDIM+1 -
```

Next NCLAS*((NDIM/2)+1) Entries; (NDIM/2)+1 entries per class

```
Element        Content
-------        -------
   1           Link Element

   2   -
       |
   .   |
   .    >      NDIM*(NDIM+1)/2 packed weighting matrix
   .   |       components (inverse of covariance matrix)
       |
NDIM+1 -
```

Remaining Entries - Number sufficient to contain NCLAS elements

```
Element        Content
-------        -------
   1           Link Element

   2   -
       |
   .   |
   .    >      NCLAS determinants of the covariance matrix of
   .   |       each class
       |
       -
```

Figure 4-17 Nearest Mean Vector Logic Block Format (continued)

Entry Number 1

Element	Content
1	Link Element
2	Number of Classes
3	Minimum Vote Count
4	Link Element to first entry containing class pair links

Next Group of Entries (logic node--data class associations) - Number sufficient to contain NCLAS+2 elements

Element	Content
1	Link Element
2	Node Number Associated w/rejects
3	Node Number Associated w/first class
.	
.	
.	Node Number Associated w/last class

Next Group of Entries - Number sufficient to contain
(class pair links)   NCLAS*(NCLAS-1)/2 elements

Element	Content
1	Link Element
2	Link Element to class pair header entry for class 1 vs. class 2
3	Link Element to class pair header entry for class 1 vs. class 3
.	
.	
.	etc.

Figure 4-18 Pairwise Logic Block Format

Remaining Entries - 3 entries per class pair

First of Three Entries (called a class pair header)

```
Element       Content
-------       -------
   1          Link Element

   2          Type of logic for pair:
                 1 - Fisher,
                 2 - Optimal Discriminant Plane,
                 3 - Any One Space,
                 4 - Any Two Space,
                 5 - Linguistic

   3          Number of thresholds or boundaries
              (function of logic type)

   4          Link Element to entry for auxiliary criteria

   5          Option number of creating or latest
              modification routine

   6          Index to first class of class pair (points to
              class name array in LI header, or to associated
              logic nodes (add 1))

   7          Index to second class of class pair (points to
              class name array in LI header, or to associated
              logic nodes (add 1))

   8 -
       \
   9    \
         \
  10      >   Five Fisher thresholds
         /
  11    /
       /
  12 -
```

Figure 4-16 Pairwise Logic Block Format (continued)

Second of Three Entries

```
Element      Content
-------      -------
   1         Link Element

   2    -
   .      \
   .       >  Coefficients of Fisher Direction
          /
NDIM+1  -
```

Third of Three Entries

```
Element      Content
-------      -------
   1         Link Element

   2    -
   .      \
   .       >  Coefficients of line perpendicular to Fisher
          /   Direction
NDIM+1  -
```

Figure 4-18 Pairwise Logic Block Format (continued)

Entry Number 1

Element	Content
1	Link Element
2	Number of Boundaries - NB (=1)
3	Number of Segments in Boundary - NS (= 1 to 5)
4	UNUSED
5	UNUSED
6	Link to Entry containing boundary segment discriminant value thresholds.
7	Index to node-number-associated-with-classes -in-the-excess-region entry (i.e., the index plus one points to node associated with the Excess Region)
8	Index to node-number-associated-with-classes -in-the-convex-region entry (i.e., the index plus one points to node associated with the Convex Region)
9	UNUSED

Remaining NS Entries

Element	Content
1	Link Element
2 . . . NDIM+1	NDIM Discriminant Coefficients

Figure 4-18a Optimal Discriminant Logic Block Format

Last Entry

Element	Content
1	Link Element
2 . . . 10	Threshold Values

Figure 4-18a Optimal Discriminant Logic Block Format (continued)

```
Entry Number 1

Element        Content
-------        -------

   1           Link Element

   2           Number of classes - NCLAS

   3           Flag:  1 - create node of overlapped vectors
                      0 - reject overlapped vectors

   4           Link element to first entry containing link
               elements to sub-blocks for each class

Next Group of Entries - Number sufficient to contain
                        NCLAS+2 elements

Element        Content
-------        -------

   1           Link element

   2 -
   .   \       NCLAS+2 list of node numbers associated
   .     >     with each of NCLAS classes, the reject
       /       node and the overlap node
     -

Next Group of Entries - Number sufficient to contain
                        2*NCLAS+1 elements

Element        Content
-------        -------

   1           Link element

   2           Link element to the last entry of the
               preceding group of entries
               (which contain the node numbers)
              --                                  --
   3         | Link element to the first entry of the |
             | sub-block for the first class          |
             |                                        |
   4         | Link element to the last entry of the  |
             | sub-block for the first class          |
              --                .                 --
   .                            .
   .
```

Figure 4-19 Closed Decision Boundary Logic Block Format

Entry Number 1

Element	Content
1	Link Element
2	Sub-block type = 1
3 . . . NDIM+2	NDIM Threshold Type Flags: 0 = user defined; 1-200 = percentage of data range (100 is the default)

Entry Number 2

Element	Content
1	Link Element
2	Basis vector type: 1-coordinate, 2-overall eigenvectors, 3-specific class eigenvectors
3 . . . NDIM+2	NDIM low threshold values

Entry Number 3

Element	Content
1	Link Element
2 . . . NDIM+1	NDIM high threshold values

Figure 4-20 Hyperrectangular Sub-block Format

Next NDIM Entries (not used if coordinate basis vectors)

```
Element      Content
-------      -------

   1         Link Element

   2    -
        |
   .    |
   .    >    NDIM components of basis vector
   .    |
        |
 NDIM+1 -
```

Figure 4-20 Hyperrectanular Sub-block Format (continued)

Entry Number 1

Element	Content
1	Link Element
2	Sub-block Type (=2)
3	Center Vector Type: 1-mean of class, 2-midrange of class, 3-user defined
4	Radius Type: 0 = user defined 1-200 = percentage of data range (100 is the default)
5	Radius (squared) of hypersphere

Entry Number 2

Element	Content
1	Link Element
2 . . . NDIM+1	NDIM components of center vector

Figure 4-21 Hypersphere Sub-block Format

Entry Number 1

Element	Content
1	Link Element
2	Sub-block type (=3)
3	Center vector type (see hypersphere sub-block for code)
4	"C" value type (see hypersphere sub-block radius type for code)
5	"C" value (corresponds to radius in hypersphere)
6	Axis Type: 0 = user-defined; 1-200 = % of default axis lengths (100 is the default)

Entry Number 2

Element	Content
1	Link Element
2 . . . NDIM+1	NDIM components of center vector

Figure 4-22 Hyperellipsoid Sub-block Format

Entry Number 3

```
Element         Content
-------         -------

   1            Link Element

   2   -
       |
   .   |
   .    >       NDIM axis lengths
   .   |
       |
NDIM+1 -
```

Remaining NDIM Entries

```
Element         Content
-------         -------

   1            Link Element

   2   -
       |
   .   |
   .    >       NDIM components of row of weighting matrix
   .   |
       |
NDIM+1 -
```

Figure 4-22 Hyperellipsoid Sub-block Format (continued)

Entry Number 1

Element	Content
1	Link Element
2	Number of characters in Boolean Statement - NC (max 132)
3	Entry Number (in LI file) for first (if more than 1) logic node to use this reject statement

Remaining Entries - Number sufficient to contain NC elements

Element	Content
1	Link Element
2 . . . NDIM+2	Characters of Boolean Statement

Figure 4-23 Independent Reject Strategy Block Format

Entry Number 1

Element	Content
1	Link element
2	Number of classes (NC)
3	K - number of nearest neighbors to be used in evaluation
4	Ignored measurements flag (=1 means there are measurements to be ignored)
5	Link to Ignored measurements vector

Entry Number 2

Element	Content
1	Link element
2 thru 9	Tree name of reference patterns

Entry Number 3

Element	Content
1	Link element
2 . . . NDIM+1	Measurements to be Ignored Vector = 1 Use Corresponding Measurement = 0 Ignore Corresponding Measurement
NDIM+2	Number of Measurements Used

Figure 4-24 Nearest Neighbor Logic Block Format

Remaining Entries - Number Sufficient to Contain NC+1 Elements

Element	Content
1	Link element
2	Node Number Associated w/first class
.	.
.	.
.	
	Node Number Associated w/last class
NC+2	Node Number Associated w/rejects

Figure 4-24 Nearest Neighbor Logic Block Format (continued)

4.8 THE LOGIC LIST (LL) FILE

The Logic List file is a list of the logic trees that exist in a user's directory. It has a two element header that contains the number of entries (logics) in the file, and an alphabetic list pointer. Each logic corresponds to an entry. An entry contains the two File Codes (FCLs) representing the files that make up a logic tree, space for an 8 character logic name, the dimension of the tree, the tree and class name pair, an incomplete logic flag, and an alphabetic list pointer. See Figure 4-25.

When a new logic is added, its entry is placed at the bottom of the LL file. When a logic is deleted, its entry is replaced by the last entry in LL and the count of the number of entries is decremented by 1.

The alphabetic list pointer, here, has the same function as the alphabetic list pointer found in the Tree List (TL) file.

4.9 THE SAVED VECTORS (SV) FILE

The SV file is a list of stored projection vectors that can be accessed by name using an arbitrary vector projection operation (S1ARBV, S2ARBV, L2ARBV). The header of the SV contains the number of vectors (see Figure 4-26). Information on each vector is stored in an entry. Vector names are one to eight characters in length, start with a letter, and are stored at the beginning of an entry followed by the dimension and the vector itself. The length of an entry is the maximum vector length allowed for normal CLPARS

AD-A118 731 PAR TECHNOLOGY CORP NEW HARTFORD NY F/G 9/2
ON-LINE PATTERN ANALYSIS AND RECOGNITION SYSTEM. OLPARS VI. PRO--ETC(U)
JUN 82 S E HAEHN, D MORRIS
UNCLASSIFIED PAR-82-15 NL

2 OF 3
AD A
118731

THE LOGIC LIST (LL) FILE

Element	Field	
1	Number of Entries	Header
2	Alphabetic List Pointer	
3	File Code (FCD) for LI File	Entry 1
4	File Code (FCD) for LV File	
5-12	Logic Name	
13	NDIM of Design Set	
14	Alphabetic List Pointer	
15 thru 26	Treename, classname of design set	
27	Incomplete Logic Flag	
	. .	
3+25(k-1)	FCD for LI	Entry k
	FCD for LV	
	Logic Name	
	NDIM of Design Set	
	Alphabetic List Pointer	
	Treename, classname of design set	
2+25k	Incomplete Logic Flag	

Figure 4-25 The Logic List File

Element		
1	Number of Vectors	Header
2	Blank	
3 thru 10	Name of Vector 1	Entry 1
11	Dimension of Vector 1	
12 thru 61	VECTOR 1	
	. . .	

Figure 4-26 The Saved Vector File

operation, which has been chosen to to be 50 for the PDP-11 systems under RSX-11M. Hence, vectors in excess measurement mode may not be stored in SV.

When a vector is deleted from the SV file, the last entry is moved up so that no "holes" appear in this file.

The command VECTOR is used to manipulate the SV file. VECTOR has five options:

1. Delete a saved vector.

2. Make a line printer listing of all saved vectors including name, length and the components of the vector.

3. Display at the screen the information in 2.

4. Display at the screen one named vector.

5. Save a user-supplied vector or the projection vector(s) used in a one or two space projection.

4.10 THE SAVED TRANSFORMATION MATRIX (SM) FILE

Under CLPARS, it is possible to save a measurement transformation for use on other data sets (see the MEASXFRM command). This option is useful for the following reason. If a logic is designed on a transformed tree, and if a test data set is run through that logic, the test data set should be first transformed in the same way as the design set. When the transformation information is stored away it can be easily "pulled out" and used on the test data set.

The transformation information is saved in the Saved Transformation Matrix (SM) File. After such commands as EIGNXFRM or NORMXFRM have completed their transformation, the programs will ask the user if (s)he wishes to save the transformation matrix. (The EIGNXFRM transformation is saved as an (NDIM)x(k) matrix, where 'k' is the number of eigenvalues chosen by the user; the NORMXFRM transformation is saved as an (NDIM)x1 matrix or vector). If the user answers yes, these routines will save the matrix in the SM file. The user can then use the saved matrix to transform another data set. The command that will accomplish this is called MATXFRM. It will prompt the user for the name of a saved matrix and then transform the current data set using the matrix (assuming that there is dimensional compatibility). The name of a matrix must be from one to eight characters in length, the first being alphabetic and the rest alphanumeric.

One additional command, MATRIX, is necessary for the maintenance of the SM file. MATRIX has the following options:

a. Delete a saved matrix;
b. Make a line printer listing of the name, type, dimension, and entries of a saved matrix;
c. Display at the screen the name type, dimension, and entries of a saved matrix;
d. Display at the screen the names, types, and dimensions of all saved matrices;
e. Save a user supplied matrix.

When a matrix is deleted, entry spaces are left in the SM file. In this case, element 2 of the SM file header will point to the first free entry and the rest of the free entry areas will be linked. The last free entry area will always occur after the last filled entry, and is pointed to by element 3 of the SM file header. The structure of the header of the SM file is pictured in Figure 4-27. MAXMTR stands for the maximum number of saved matrices the user may have. MAXMTR set equal to 10 should be more than sufficient.

An entry in the SM file (Figure 4-28) will be the length of the maximum feature vector allowed in CLPARS, without going into excess measurement mode (which is 50 for the PDP-11/70 under RSX-11M) plus one to allow for a link pointer. For NORMXFRM, one entry will suffice to store the (NDIM)x1 transformation matrix. For EIGNXFRM, the transformation matrix will be stored in as many entries as necessary.

	Element	
	1	Number of Saved Matrices
	2	Pointer to the Head of the Entry Free List
	3	Pointer to the End of the Entry Free List
	4	Number of Vectors in File
Description of Matrix "1"	5 through 12	Name of Saved Matrix "1"
	13	Type *
	14	NDIM
	15	Entry # of start of Matrix **
		. . .
Description of Matrix MAXMTR		Name of Saved Matrix "MAXMTR"
		Type
		NDIM
	4+(11*MAXMTR)	Entry No. of Beginning of Matrix

* NORMXFRM = -1
EIGNXFRM = R, where R is the no. of eigenvalues used.

** 0 implies not in use

Figure 4-27 The Saved Transformation Matrix File Header

Element	
1	Link Pointer to Next Vector in Matrix *
2 thru NDIM+1	V E C T O R

* 0 implies last vector (row) of matrix

Figure 4-28 The Saved Transformation Matrix File Entry

SECTION 5

DISPLAYS

5.0 INTRODUCTION

There are six types of displays that CLPARS can produce: two-space cluster plots, two-space scatter plots, rank order displays, confusion matrices, one-space macro plots, and one-space micro plots. Since the information to create each of these types of displays is so different, the structure of the display files will be discussed individually by type of display. However, each type uses (at least) two files: the Display Information (DI) file and the Display Value (DV) file. The DI file will contain necessary controlling information and pointers into the DV file, and the DV file will contain values to be displayed. In general, the values to be displayed are saved so that the process of redisplay (that is, recreating the last display on the terminal) is a simple one, and display manipulations (e.g., elimination or addition of a class on a two-space display) does not require a complete regeneration of the display.

The first element in the LI file will always be a display code as follows:

0 = not set

1 = two-space cluster

2 = two-space scatter

3 = rank order

4 = confusion matrix

5 = one-space macro

6 = one-space micro

5.1 TWO-SPACE DISPLAYS (SCATTER AND CLUSTER)

A two-space scatter plot is a two-dimensional representation of an n-space vector, with each vector located at its "natural" projection point on the screen.

A two-space cluster plot is a two-dimensional representation of an n-space vector, with each vector "forced" into a location within a grid. If one or more vectors from a single data class fall within the same grid location, the display symbol for that class is presented. If vectors from two or more data classes fall within a single grid location, an asterisk is displayed.

The actual presentation of the two-space cluster display is generally faster than the two-space scatter display, especially for a large data set. However, since each character displayed may represent one or more vectors, in some cases this display could be

misleading.

5.1.1 THE DISPLAY INFORMATION (DI) FILE -

The display file set up for two-space displays allows the CLPARS applications programs to efficiently create a new display, view an old display, or change the scale of two-space plots. The DI file header (see Figure 5-1) contains the following information.

- The display code (element 1).
- A code (element 2) for the type of coordinate projection used: L1, L2, S1, S2 versions of ASDG, CRDV, EIGV, FSHP; or NLM.
- The treename, nodename pair (elements 3 through 14) that makes up the (current) data set for which the two-space plot was made.
- The dimension of the data set (element 15) and the number of lowest nodes in that data set (element 16).
- The number of boundaries is in element 17. There is a maximum of 2 boundaries on any one or two-space plot.
- Elements 18 and 19 enable CLPARS programs to access the boundary points (and 2-space convex point) of the first

Element		
1	Display Code	
2	CLPARS Option number	Projection type
3-14	DATA SET name	
15	NDIM	
16	Number of Classes	Number of lowest nodes in the data set
17	Number boundaries	
18	No. of points in 1st boundary	
19	First boundary pointer	Points to (x,y) coor.s. of the boundary in DY file
20	No. of points in 2nd boundary	
21	Second boundary pointer	
22-23 24-25	Orig. x(min), x(max) y(min), y(max)	
26-27 28-29	Current x(min), x(max) y(min), y(max)	
30	1st proj. vect. number (ptr.)	
31	2nd proj. vect. number (ptr.)	
32-39	LOGIC NAME	
40	Logic Node Number	
41	Total Number of Vectors	
42	Number of Bins (NB)	
43	Type of Scaling (1-space) (2-space) 0 = probability rectangular 1 = counts square	
44	Zoom Flag (0 = not zoomed) (1 = zoomed)	
45	Intensity Flag (for one-space micro only)	(0 = no class has been intensified) (1 = a class has been intensified)

Figure 5-1 The Display Information File Header for One-Space and Two-Space Displays

boundary in the LV file. (The convex point is found in the entry immediately following the last point of each boundary. The number of points in a boundary (elements 18,19) does not include the convex point).

- Elements 20 and 21 provide the same information for the second boundary, as do elements 18 and 19 for the first boundary.

- Elements 22 through 25 contain the minimum and maximum x, y values for the projected vectors.

- Elements 26 through 29 contain the minimum and maximum values that are used to create the two-space plot. These values are initially the same as those in elements 22 through 25. When a scale change is made by the user, using the command SCALZM (scale zoom), the screen coordinates that specify the new scale are transformed into the coordinate system of the projection vectors, and are stored as the current minimum and maximum values. Vectors that lie outside of this new region are not displayed on the screen.

- Elements 30 and 31 contain pointers to the projection vectors (in the PROJECTION VECTOR file) used.

o Elements 32 through 40 are used in the logic one-space (L1) and two-space (L2) projection routines. They store the name of the logic and the node number for which the projection was made.

o Element 41 contains the total number of vectors in the data set. It is used in determining whether to produce a scatter or cluster plot. If it is larger than 500, a cluster plot is produced. If it is 500 or less, a scatter plot is produced. This default value of 500 may be changed by the user (see the command CDEFAULT).

o Elements 42 and 45 are for one-space displays only and are ignored in the two-space case.

o Element 43 determines the type of scaling to be used for a two-space display, either square or rectangular. Square scaling is the default and may be changed by the the user (see the command CSCALE).

o Element 44 is the zoom flag for one or two space displays and when set, indicates zooming is in effect (i.e., the display scale has been modified to get a "closer" look at classes found on the display).

For each class (lowest node in the data set) there is a logical entry in the DI file (see Figure 5-2).

All vectors from a class will be stored in the DV file in sequence. Therefore, elements 5 and 6 of the logical entry for that class will give access to all of the projected vectors. (Element 6 actually points to the projection of the mean vector of the class.) Element 7 indicates whether or not the vectors from the class should be displayed on the screen. This element is initially "on" and can be reset by using the command SELECT. The display flag symbol (element 8) appears next to the class symbol of "classes-displayed-and-currently selected" list of the one or two space display. (Currently, this symbol is only important with displays using the Fisher discriminant plane. All projection subroutines (e.g. DSPROJ, LCPROJ) initially make this symbol a 'blank.')

Element	
1-4	CLASS NAME
5	number of vectors in the class
6	Pointer to projected vectors in DV
7	Display flag (1 means display class 0 means don't show it)
8	Display Flag Symbol

Figure 5-2 The DI File Logical Entry for Two-Space Displays

5.1.2 THE DISPLAY VALUE (DV) FILE -

The general structure of the DV file for a data set with 'K' lowest nodes is pictured in Figure 5-3.

The DV file header (see Figure 5-4) is just four elements long. The first two elements contain information for the purpose of cross checking the DV file with the DI file. The third element indicates whether or not screen coordinates have been computed. This flag will be used in situations where vectors must be redisplayed but the screen coordinates have already been computed (e.g., the commands SELECT, RDISPLAY). Element 4 points to the next available entry at the end of the file.

The projected vectors from a data class are stored together in a block in the DV file (Figure 5-3). The first vector in each block is always the projection of the mean vector of the class. For structure analysis displays, the mean vector of a class is stored in the TI file and is projected and inserted into the DV file before the rest of the vectors from that class are projected and stored. For logic design displays, the projected mean vector may have to be computed after the rest of the vectors from a class (or part of a class) are projected.

In either situation, the mean vectors will not be initially displayed on the terminal screen. If users wish to see the projected mean vectors (for the classes that are being displayed) superimposed on the screen, they should invoke the command PROJMN.

```
                   --------------------------
                   | Header                 |
                   |------------------------|
                   | Projected              |
                   | Vectors from           |
                   | Class 1                |
                   |------------------------|
                   | Projected              |
                   | Vectors from           |
                   | Class 2                |
                   |------------------------|
                   |                        |
                             .
                             .
                   |                        |
                   |------------------------|
                   | Projected              |
                   | Vectors from           |
                   | Class k                |
      ----------   |------------------------|
                   | Boundary points        |
                   | from boundary 1        |
      Optional     |------------------------|
                   | Boundary points        |
                   | from boundary 2        |
      ----------   --------------------------
```

Figure 5-3 The DV File for One- and Two-Space Displays

Element		
1	Display Code	
2	OLPARS Option Number	
3	Screen Coordinate Flag	0=screen coordinates not computed 1=screen coordinates computed
4	Next Available Entry	

Figure 5-4 The DV File Header for One-Space and Two-Space Displays

PROJMN will display the mean vectors on the existing display with small rectangles around the appropriate class symbol

A DV file logical entry (see Figure 5-5) contains all the necessary information about a projected vector or a boundary point. The ID of a mean vector is a negative one (-1) and the ID of a boundary point is zero (0). Screen coordinates are discussed in Section 5.1.4.

When storing boundary information, the actual boundary points will be stored first (there can be at most six such points), followed by the point on the "convex side" of the boundary. The x and y coordinates of the boundary points will be computed by DRAWBNDY as they are entered by the user.

5.1.3 THE PROJECTION VECTOR (PV) FILE -

There is one other file that will be necessary for the efficient display of two space plots. This is the Projection Vector (PV) File. It contains the n-dimensional vectors on which the data set is projected. Each entry is of length NDIM and contains a projection vector. In addition, eigenvectors (and eigenvalues) that have not been chosen by the user, are saved in this file. If the user wishes to reproject on a different set of eigenvectors, for example, they are immediately available for use. The structure of the PV file is pictured in Figure 5-6. Notice that the vectors actually used in the projection are pointed to in

Element		
1	Vector I.D.	0 if boundary point -1 if mean vector
2	x coordinate	
3	x screen coordinate	
4	y coordinate	
5	y screen coordinate	

Figure 5-5 The DV File Logical Entry for Two-Space Displays

	Element	
Header	1	Display Code
	2 thru 13	Current Data Set
	14	NDIM
	15	Number of measurements used
	16	Number of projection vectors
	17	Pointer to 1st projection vector
	18	Pointer to 2nd projection vector
Entry 1		Measurement vector
Entry 2		First Vector
Entry 3		Second Vector
. . .		
Entry K+1		Kth vector
Entry K+2		Eigenvalues (only for ..EIGV or ..GNDV commands)

Figure 5-6 The PV file for One and Two-Space Displays

elements 17 and 18 of this file. These vectors are also referenced in elements 30 and 31 of the DI file header. For coordinate projections, the PV file contains an identity matrix. For the ASLC and FSHP commands this file contains only one or two vectors.

FORTRAN OLPARS allows users to delete specific measurments in most structure analysis or logic design projections. Any measurements that are deleted from such a projection are kept track of in the PV file as follows. The number of measurements used in the computation is saved in element 15. The first entry of the PV file contains a measurement vector (of length NDIM) which is a vector of zeroes and ones. A one occurs if the measurement is used in the computation; otherwise, a zero is placed in that coordinate. For the eigenvalue options, the number of eigenvalues equals the number of measurements used.

The projection vectors stored in the PV file also have length NDIM. This results from placing zeroes in the deleted coordinate positions of the projection vectors.

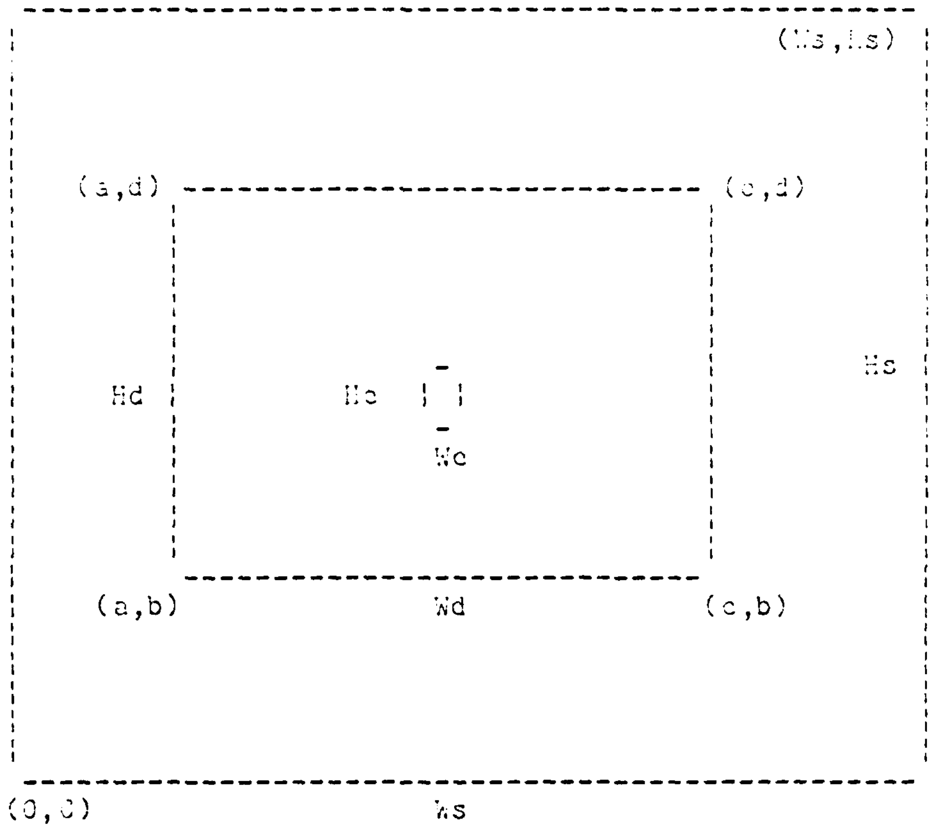
5.1.4 Screen Coordinates -

The set of x and y "coordinates", found in the DV file logical entry (see Figure 5-5) for each vector, is the projection point of the vector onto the plane of the projection vectors. This set of coordinates remains fixed as long as the projection vectors remain fixed. The set of x and y "screen coordinates" is the system dependent screen position of the point. The "screen coordinates" depend on which calculated two space option is used, cluster or scatter.

A hypothetical screen for CLPARS with a rectangular two-space display area is pictured in Figure 5-7. It is assumed that:

a. The coordinates start at (0,0) in the lower left corner of the screen.

b. Each character is represented in a rectangle Wc units by Hc units and is displayed by specifying coordinates for its lower left corner.



Ws = number of display units in screen width
Hs = number of display units in screen height
Wd = number of display units in display width = c-a
Hd = number of display units in display height = d-b
Wc = number of display units in character width
Hc = number of display units in character height

(a,b), (c,b), (c,d), and (a,d) are the screen coordinates of the display rectangle.

Figure 5-7 A hypothetical screen

For a scatter plot, the screen coordinates (Xs, Ys) of a vector are given by

$$
Xs = \begin{cases} \dfrac{X - Xmin}{MAX}(Wd - 2Wc) + a + Wc & Xmin <= X <= Xmax \\[2ex] 0 & X < Xmin \text{ or } X > Xmax \end{cases}
$$

$$
Ys = \begin{cases} \dfrac{Y - Ymin}{Ymax - Ymin}(Hd - 2Hc) + b + Hc & Ymin <= Y <= Ymax \\[2ex] 0 & Y < Ymin \text{ or } Y > Ymax \end{cases}
$$

where x,y are the projected values of the vector and

MAX = maximum (Xmax - Xmin, Ymax - Ymin)

when 'square' scaling is used and

MAX = (Xmax - Xmin) for Xs
MAX = (Ymax - Ymin) for Ys

when 'rectangular' scaling is used.

The "current values" of Xmin, Xmax, Ymin, Ymax (found in the DI file header) are used and Xs, Ys are interpreted as integer values for display. The calculation of (Xs, Ys) is set up so a display point never falls on the display border.

For cluster plots, a grid must be superimposed on the OLPARS display window (see Figure 5-8). We now interpret the screen coordinates (Xs, Ys) of a vector to be the cell in which the projected vector (x,y) lies.

Xs and Ys are given by the formulas:

$$
Xs = \begin{cases} \dfrac{X - Xmin}{MAX}(M) + 1 & Xmin <= X < Xmax \\ M & X = Xmax \\ 0 & X < Xmin \text{ or } X > Xmax \end{cases}
$$

$$
Ys = \begin{cases} \dfrac{Ymax - Y}{MAX}(M) + 1 & Ymin < Y <= Ymax \\ M & Y = Ymin \\ 0 & Y < Ymin \text{ or } Y > Ymax \end{cases}
$$

Again, Xs, Ys are interpreted as integer values. Note, for both cluster and scatter displays, if Xs = 0 or Ys = 0, it means that the point (Xs, Ys) fell outside of the region to be displayed. Such points are not displayed.

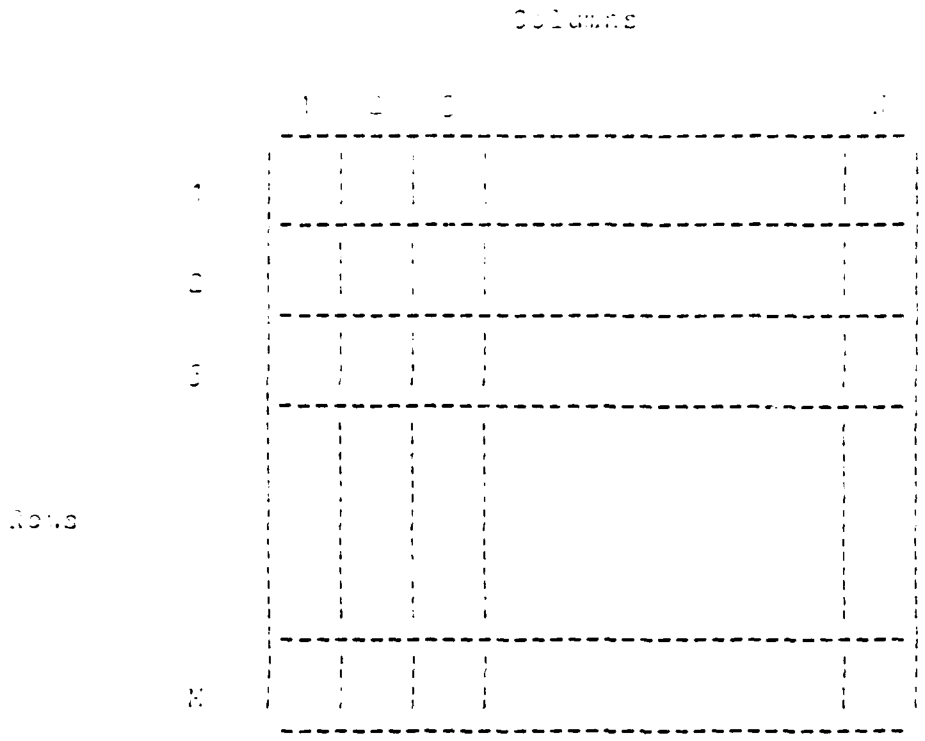


Figure 5-8 Display Rectangle with Cluster Plot Grid

Some additional considerations concerning screen coordinates are as follows:

- o The constants Ws, Hs, Wd, Hd, Wc, Hc, a, b, c, d, M, and N are all system dependent and are determined by experimentation with the particular terminal. In order to make much of the code that utilizes these constants system independent, we did the following:

  - oo The constants reside in an CLPARS file that can be easily updated by "systems" personnel.

  - oo One of the functions that the HELCLP command provides is to request the terminal type from the user. HELCLP will then place the correct constants into the CM file (see Section 4.1) for use by the display programs. This way the same CLPARS programs can put up displays on different size terminal screens.

Appendix B shows an example file containing screen coordinates for a Tektronix 4051 terminal.

5.1.5 Scaling In Two-Space Displays -

The 'type of scaling' flag and 'zoom' flag (elements 43 and 44 of the two-space display information file header) control the scaling found on the two-space display.

Initially, all two-space projections have 'square' scaling. This means that the value of the measurement units on both the x and y axes are equal. When scale "zooming" (obtaining a close-up view of a subsection of the original display) or a scale change occurs, the scaling becomes 'rectangular', that is, the value of the measurement units on the x and y axes are no longer equal.

5.1.6 Original And Current Min-Max Coordinates -

The 'original' minimum and maximum coordinates are obtained during initial data projection computations (Global Scale). Since the initial two-space scaling is 'square', the 'current' coordinates are obtained by using the maximum range of the 'original' min-max coordinates. Thus, the current min-max coordinates are readjusted original coordinates. If the CSCALE (change scale) command is used immediately after a two-space data projection, the display would use 'rectangular' scaling with the current min-max coordinates equal to the original min-max coordinates. If the SCALZM (scale zoom) command is used to obtain a display subsection, the current min-max coordinates reflect the exact subsection requested by the user (zoom scale). If the CSCALE command is used after SCALZM, the maximum range of the current

min-max coordinates is used to readjust those coordinates so that the type of scaling is 'square'. Where returning the display back to its global scale (SCALRET command), the 'square' scaling range is used again for the current min-max coordinates. See Figure 5-CA for a table summary of the original and current min-max coordinate states.

5.2 ONE-SPACE DISPLAYS (MICRO AND MACRO)

CLPARS can produce two types of one space displays. They are referred to as "micro" and "macro" displays. The one-space micro display is a view of selected data class histograms presented in symbolic format. Using the INTENSIFY command, selected class histograms can appear as bar graphs. The one-space macro display is a view of selected data classes in a "stack histogram" format. Examples of these types of displays appear in the CLPARS VI User's Manual.

The display files for one-space displays are very similar to those for two space displays. The DI file header, shown in Figure 5-1, is identical to the two space case. Information like Ymin, Ymax, number of points in a boundary, that are only used in two space displays, are ignored. Element 42 contains the number of bins used in the display; element 43 is used to determine the vertical scale for micro plots; element 45 shows whether or not any of the classes for a micro-view display have been "intensified" (micro-plots only).

Global Scale (initial scale type is 'square')

	square	rectangular
original min-max coordinates	min-max data values	min-max data values
current min-max coordinates	readjusted* originals	min-max data values

Zoom Scale (initial scale type is 'rectangular')

	square	rectangular
original min-max coordinates	min-max data values	min-max data values
current min-max coordinates	readjusted* zoomed values	zoomed data values

* readjusted - maximum coordinate range used

Figure 5-8a Original and Current Min-Max Coordinate States

The DI file logical entry one space displays is shown in Figure 5-9. The intensity flag is used to determine whether histograms should be drawn on one space micro displays. For each class in a data set or logic node, there will be a logical entry in the DI file. The histogram information for each class is stored in the scratch 1 (S1) file and can be accessed via the bin count pointer of the 1-space logical entry.

The LV file for one space displays has the same header as for two space displays (see Figure 5-4), and a logical entry, one for each projected vector or boundary point, as pictured in Figure 5-10. In order to define the "screen coordinate" of a vector in a one space display, let NB be the number of bins for the display. Then, NB is given by the equation

$$NB = \min\left(\frac{\text{total \# of vectors}}{(\text{\# of classes})(\text{one-space bin factor})}, MB\right),$$

where MB is the maximum number of bins allowed on a one space display. The one space bin factor has the default value of 5, as in MCCS, but can be changed by the user executing the command CDEFAULT (this bin factor is stored in the CM file).

Element	
1	C L
2	A S
3	S N A
4	M E
5	number of vectors in class
6	Ptr. to projected vectors in DV
7	Display flag (1 means display class 0 means don't show it)
8	Display Flag Symbol
9	Intensify flag (1 means intensify 0 means don't do it)
10	Ptr. to bin counts in S1

Figure 5-9 The DI File Logical Entry for One-Space Displays

Entry		
1	Vector ID	0 if a boundary point -1 if mean vector
2	x coordinate	
3	x screen coordinate	

Figure 5-10 The DV File Logical Entry for One-Space Displays

Xscreen is defined by the equation

$$\text{Xscreen} = \begin{cases} \dfrac{X - Xmin}{Xmax - Xmin}(NB) + 1 & \text{if } Xmin <= X < Xmax \\ NB & \text{if } X = Xmax, \\ 0 & \text{if } X < Xmin \text{ or } X > Xmax \end{cases}$$

where the current values of Xmin and Xmax are used, and Xscreen is interpreted as an integer. Xscreen is the bin number in which the projected vector lies.

The PV file for one space is the same as for two space (see Figure 5-6), except that element 18 is ignored.

One space displays use one additional file; the S1 (Scratch 1) file. For each class in the data set or logic node, S1 will contain the number of vectors (from that class) in each bin. Storing this additional information will require less computations in options like REDISPLAY and SELECT.

The S1 file header (see Figure 5-11) contains information that enables programs to decide whether or not the S1 file contains the correct data. The S1 header also contains the maximum bin count (over all classes) which is used in determining the length of the bars in the one space macro display. The length of a logical entry in S1 will be the maximum number of bins allowed. There is one

logical entry for each class and this entry is pointed to by element 9 of the DI file entry for that class. Figure 5-12 shows the format of the S1 logical entry for one-space displays.

5.2.1 Screen Parameters For One Space Displays -

The following set of system dependent screen parameters, that will be accessible to CLPARS in the way described in Section 5.1.4., are necessary for one-space displays (see Figure 5-13).

(e,f) - The screen coordinates of the display line of
(g,f) the first class displayed. (Classes are displayed from the bottom, upwards).

Ds - The distance between successive display lines.

NC - The number of classes that can be displayed on the screen at one time in a macro display. The coordinates of the endpoints of the last display line are the (e, f + (NC - 1)Ds) and (g, f + (NC - 1)Ds).

The heights of the bars in the one space macro display is computed as follows. The bar representing the maximum bin count is based on the value Ds-6. All of the other bars are then scaled according to this height. See the subprogram MACROP for further information.

Element	
1	Display code
2	OLPARS option number
3	Number of bins
4	Maximum bin count (No. of Vectors in fullest bin)

Figure 5-11 The S1 File Header for One-Space Displays

Element	
1	Number of vectors in bin 1 *
2	Number of vectors in bin 2 *
	.
	.
	.
NB	Number of vectors in bin NB *
NB+1	Blank
	.
	.
	.
Max No. of bins	Blank

* If probabilities are used instead of counts in a one-space micro display, these probabilities are computed by the program MICROP but are not stored in S1.

Figure 5-12 The S1 File Logical Entry for One-Space Displays

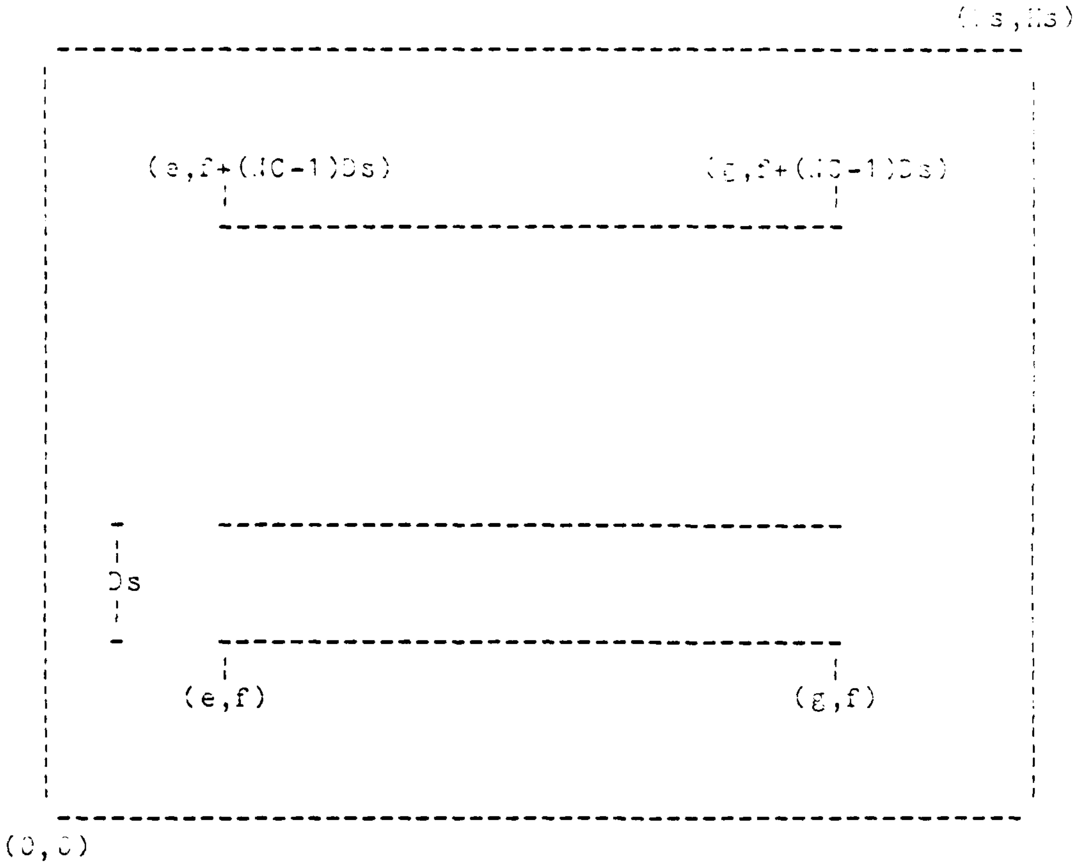


Figure 5-13 One-Space Display parameters

5.3 DISPLAY FILES USED IN MEASUREMENT EVALUATION COMMANDS

The Display Information (DI) file and the Display Value (DV) file, where used with Measurement Evaluation Commands, have formats as described below. The general scheme is similar to display handling for other CLPARS functional applications in that the DV file contains the basic, raw data computed by the major analytic function (Discriminant Measure or Probability of Confusion); the DI file contains control information and the formatted or manipulated subset of the raw data, organized for display to the user, as determined by the associated subsidiary commands.

In the case of PROBCONF, an extra file, S1 (Scratch 1), is used (temporarily) to support computation of values which are stored in the DV file.

Formats for these files are given in the following figures:

Figure 5-14	Rank Order DI Header
Figure 5-15	Rank Order DI Entry
Figure 5-16	Rank Order DV File
Figure 5-17	Rank Order S1 Header (PROBCONF Only)
Figure 5-18	Rank Order S1 Entries (PROBCONF Only)
Figure 5-19	Rank Order S1 Example (PROBCONF Only)

Element	Contents	
1	display code (=[illegible])	
2	creating command option no.	
3 - 14	current data set name	
15	ndim - number of measurements	
16	nclas - number of classes	
17	rank option number	
18 - 19	rank option parameters (See Table Below)	
20	sort order flag 0=descending 1=ascending	
21 thru 20+MAXCLS	ordered list of class display characters	
+1	asterisk flag/index to best class	for meas. no. 1
+2	indices to best class pair	
.		
.		
.		
	asterisk flag/index to best class	meas. no. EXMLIM
+2*EXMLIM	indices to best class pair	

Type of Ranking	Option No. (ele. 17)	Option Parameters (elements 18 and 19)	
overall	1	--	--
meas. by class	2	class symbol	
meas. by class pr.	3	1st class symbol	2nd class symbol
class by meas.	4	meas. number	--
class pr. by meas.	5	meas. number	--

The asterisk flag is the algebraic sign; (+) indicates not set, asterisk off; (-) indicates set, asterisk on. Indices to best class pair are packed (into one real element) by the scheme 100*firstclass index + second class index. (Packing only works for class indices less than 3 decimal digits long.)

Figure 5-14 Rank Order DI Header

Element	Contents		
1	value	Ranked values and associated indices	Entry 1
2	index(indices)		
3	value		
4	index(indices)		
.			
.			
.			

In entry 1 (the only entry) The index element is a function of the rank option number (it is a measurement no. if the option number = 1, 2, or 3; a class index if option number = 4; or a class pair indices (as in the header) if option number = 5).

Number of elements in logical entry (LENE) = NCLAS*(NCLAS-1)/2, where NCLAS represents the number of classes in the data set.

Figure 5-15 Rank Order DI Entry

```
 Element                  Contents
                -----------------------------  --------
     1          | Display code (=3)           |
                |-----------------------------|  Header
     2          | creating command opt. no.   |
                |-----------------------------|  --------
     3   "A"    | D(1,2) [x(1)],...[x(ndim)]  |  Entry 1
                |-----------------------------|  --------
 3+ndim+1       | D(1,3) [x(1)],...[x(ndim)]  |
                |            .                |  Entry 2
                |            .                |
                  D(nclas-1,nclas) [x(1)]        --------
                |            .                |  Entry S =
                |            .                |  (nclas*(nclas-1))/2
                | D(nclas-1,nclas) [x(ndim)]  |
                |-----------------------------|  --------
        "B"     | D(1) [x(1)],...[x(ndim)]    |
                |                             |  Entry S+1
                |                             |
                |                             |  --------
                             .
                             .
                |            .                |  --------
                |                             |
                |                             |  S+nclass
                | D(nclas) [x(1)]...[x(ndim)] |
                |                             |
                |-----------------------------|  --------
        "C"     | D[x(1)],...D[x(dim)]        |  S+nclass+1
                -----------------------------    --------
```

Length of header (HNE) = 2
Length of Logical Entry (LENE = ndim)

Order of classes is the same as the
ordered list of class display characters
in the DI file header.

"A" - measurement discr. for class pair i,j on meas. x(k)
"B" - measurement discrimination for class i on meas. x(k)
"C" - measurement discrimination for measurement x(k)

Figure 5-16 Rank Order DV File

Element	Contents
1	Display code (=3)
2	Creating command opt. no.
3	No. of measurements (ndim)
4	No. of classes (nclas)
5	total no. of cells for each class
6	NC(1) no. of histogram cells meas. 1
7	OP(1) offset ptr. for cell cnt. meas. 1 =1
8	CW(1) cell width meas. 1
9	MIN(1) min. value meas. 1
10	MAX(1) max. value meas. 1
11	NC(2)
12	OP(2) = NC(1) + OP(1)
13	CW(2)
14	MIN(2)
15	MAX(2)
	.
	.
	NC(NDIM)
	OP(NDIM) = NC(NDIM-1) + OP(NDIM-1)
	CW(NDIM)
	MIN(NDIM)
5*ndim+5	MAX(NDIM)

.

Length of Header (HNE) = 5+(5* ndim)*nclas

Figure 5-17 Rank Order S1 Header

Element	Contents
OP(j)	count for cell 1 class i, meas. j
OP(j)+1	count for cell 2 class i, meas. j
OP(j)+2	count for cell 3 class i, meas. j
OP(j)+ NC(j)-1	count for cell NC(j) class i, meas. j+1
OP(j+1)	count for cell 1 class i, meas. j+1
OP(j+1)+ NC(j+1)-1	count for cell NC(j+1) class i, meas. j+1

Order of entries is:

class 1 meas. 1
class 1 meas. 2

.
.

class 1 meas. ndim
class 2 meas. 1
class 2 meas. 2

.
.

class nclas. meas. ndim

Length of Entry (LENE) = 1

Figure 5-18 Rank Order S1 Entry

```
HEADER                             ENTRIES
------                             -------

Display code(=3)               1) cell 1, meas. 1 --
ndim(=5)                       2) cell 2, meas. 1   |
nclas(=4)                      3) cell 3, meas. 1   |
total cells(=15)               4) cell 1, meas. 2   |
-------------                  5) cell 2, meas. 2   |
NC(1)=3                        6) cell 3, meas. 2   |
OP(1)=1 ...                    7) cell 4, meas. 2   |
-------------                  8) cell 1, meas. 3    > class 1
NC(2)=4                        9) cell 2, meas. 3   |
OP(2)=4 ...                   10) cell 1, meas. 4   |
-------------                 11) cell 2, meas. 4   |
NC(3)=2                       12) cell 3, meas. 4   |
OP(3)=8 ...                   13) cell 1, meas. 5   |
-------------                 14) cell 2, meas. 5   |
NC(4)=3                       15) cell 3, meas. 5 --
OP(4)=10 ...
-------------                          .
NC(5)=3                                .
OP(5)=13 ...
-------------
     .
     .
```

Figure 5-19 Example of Rank-order S1 file offsets and pointers

5.4 CONFUSION MATRICES

There are two types of confusion matrices that CLPARS can produce; between-group and within-group. The between-group confusion matrix (see Figure 5-20) is generated when designing logic using a one-space or two-space group logic command, partitioning the resultant data projection, and finally using CREATLOG to create the group-logic. Since it might be desirable to redisplay the one or two-space projection and the boundaries upon which the logic might have been based, this type of confusion matrix is not stored in the display files.

The within-group confusion matrix (see Figure 5-21) is produced as a result of the design or application of a complete within-group logic (closed decision boundary, Fisher, NMV), a logic evaluation on the design set, or a logic evaluation on a test set. The information to create this confusion matrix is stored in the DI file. The DI file structure for the within-group confusion matrix is described next.

```
CONFUSION MATRIX (BTWN. GROUP LOGIC)  DATE: 22-JAN-82   12:24:24

  REGION       LOGIC  DISPLAY SYMBOLS OF   LOGIC NAME- grains
               NODE   ASSOCIATED CLASSES

CONVEX (1)      4     w
CONVEX (2)      3     scor
EXCESS          2     aC

        L O G I C   N O D E S             SUMS AND PERCENTAGES
CLASS  (PARENT)    (CHILDREN)              (CORRECT)    (ERROR)
NAMES     1          4       3       2    COUNT PRCNT  COUNT PRCNT
-----                                     ------------------------
soy      42          0      42       0      42 100.0      0   0.0
corn     42          0      42       0      42 100.0      0   0.0
oats     45          0      45       0      45 100.0      0   0.0
weat     42         41       1       0      41  97.6      1   2.4
Clov     43          0       0      43      43 100.0      0   0.0
alfa     37          0       0      37      37 100.0      0   0.0
rye      42          0      42       0      42 100.0      0   0.0

TOTAL   293         41     172      80     292  99.7      1   0.3
CORRECT             41     171      80
(PRCNT)          100.0    99.4   100.0
ERRORS               0       1       0
(PRCNT)            0.0     0.6     0.0
```

Figure 5-20 Between-Group Confusion Matrix (not stored in a file)

PARTIAL NEAREST MEAN VECTOR EVALUATION FOR LOGIC NODE 1

WITH EUCLIDEAN DISTANCE OPTION
LOGIC NAME - grainlog
DESIGN DATA SET - grains (****)
DIMENSIONALITY - 12

ASSIGNED CLASSES

	soy	corn	oats	weat	Clov	alfa	rye	RJCT
soy	19	0	0	0	0	0	0	0
corn	1	17	0	0	0	0	0	0
oats	0	0	20	0	0	0	0	0
weat	0	0	0	18	0	0	0	0
Clov	0	0	0	0	14	5	0	0
alfa	0	0	0	0	1	16	0	0
rye	0	0	1	0	0	0	17	0

TRUE CLASS	soy	corn	oats	weat	Clov	alfa	rye
TOTL	19	18	20	18	19	17	18
CORR	19	17	20	18	14	16	17
PRCT	100.0	94.4	100.0	100.0	73.7	94.1	94.4
ERCR	0	1	0	0	5	1	1
PRCT	0.0	5.6	0.0	0.0	26.3	5.9	5.6
REJT	0	0	0	0	0	0	0
PRCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TOTAL NUMBER OF VECTORS = 129
OVERALL CORRECT 121 FOR 93.80 PRCT
OVERALL ERROR 8 FOR 6.20 PRCT
OVERALL REJECT 0 FOR 0.00 PRCT

Figure 5-21 Within-Group Confusion Matrix

The DI file header (see Figure 5-22) contains information summarizing the logic evaluation, as well as the names of the assigned classes, i.e., those classes that reside at the lowest logic nodes. The constant, Max, denotes the largest number of data classes that a logic tree may have, which is assumed to be 50.

For each true class, i.e., the data class being evaluated, there is an entry in the DI file. It contains the class name and the information that appears under the name in the confusion matrix display. See Figure 5-23.

The DV file is not used to create the confusion matrix display at the screen. The DV file may be used at a later date to store certain logic design error information that can be sent to the printer under the various line printer options. However, this is presently accomplished by creating a temporary sequential file to store such information.

Element	
1	Display Code (=4)
2	OLPARS option number
3 thru 14	DESIGN DATA SET NAME
15	Number of true classes
16	NDIM
17	Number of vectors
18 thru 25	LOGIC NAME
26	Logic node number -- (0 means file was created thru an overall evaluation)
27	Total number correct
28	Total number errors
29	Total number reject
30	Percent correct
31	Percent errors
32	Percent reject
33	Number of assigned classes
34	Logic type (-1 for overall evaluation)
35-38	Name of assigned class 1
39-42	Name of assigned class 2
	.
	.
	Name of last assigned class
34+4*Max	Blank

Figure 5-22 The DI File Header for Confusion Matrix Display

1	True
.	Class
.	Name
4	
5	Number of Vectors
6	Number Correct
7	Number Errors
8	Number Reject
9	Percent Correct
10	Percent Error
11	Percent Reject
12	Number in first assigned class
.	Number in second assigned class
.	.
.	.
11 + number of assigned classes	Number in last assigned class
11 + Max	

Figure 5-23 The DI Logical Entry for Confusion Matrix Displays

SECTION 6

TERMINAL AND TEXT FILE INPUT/OUTPUT

6.0 INTRODUCTION

The following sections describe the terminal character I/O package, the terminal graphics I/O package, and the text file I/O package. The terminal character I/O and the text file I/O packages were based on similar packages that can be found in the UNIX I/O library. The graphics I/O package resembles the FORTRAN PLTMAP routines.

6.1 CLPARS TERMINAL CHARACTER INPUT/OUTPUT

The CLPARS programs use terminal input and output character handling routines. Due to the nature of the task required of these routines, they should be considered system dependent.

The terminal input routine (TRMGET) obtains its information as a character string. It will perform a translation of these characters to some internal representation, specified by a format control string. The reason for having a terminal input routine translate the user input characters and not a FORTRAN I/O package, is that some FORTRAN I/O packages abort a program when the input cannot be interpreted properly. This is totally unacceptable for the CLPARS system. By having its own terminal input routine, an

CLPARS program can control what happens when it receives invalid input.

To be consistent and not rely on what a FORTRAN I/O package can handle, CLPARS programs also use a terminal output routine (TRMPUT). This routine translates the internal representation of data into a character representation, via a format control string.

The following text explains in greater detail what a format control string may contain, and how the terminal I/O routines will function. Most of the format conversion characters coincide with FORTRAN's conversion characters. The specific actions taken during a conversion process, however, may be slightly different.

To be able to format terminal output, it is necessary to have complete and easy control over the terminal writing mechanism (i.e., cursor, type ball and carriage, matrix printer head, etc.).

In this section, the writing mechanism will be known as "cursor".

To control the cursor, a programmer needs to specify when it should move down a line (line feed), or when it should move to the right (tabs, spaces) or left (backspaces, return). The following paragraphs specify a character representation of the cursor control, that will be used in the terminal I/O format control strings.

6.1.1 Special Characters Within A Format Control String -

The ':' provides for writing tabs, new lines, backspaces, line feeds, and carriage returns so that they are visible to a programmer. The symbol ':' is known as an "escape character", i.e., whatever character follows ':' is in some way special.

The previously mentioned special characters are represented as follows.

:N	newline character, i.e. carriage return and line feed
:B[n]	backspace 'n' character positions
:L	line feed character
:R	carriage return character
:T	tab character (tab stops are placed at eight character intervals from beginning of line, i.e., 1, 9, 17, 25 ...)
:F	form feed character

To obtain the escape character in a string, '::' must be used. (Note, due to FORTRAN conventions, the single quote character cannot be "escaped".)

Other special character meanings:

:P[n]	place line cursor to 'n'th character position within the current (buffer) line.
:X[n]	input: skip over the next 'n' characters in the current (buffer) line output: transmit 'n' blanks to the current (buffer) line. Note, if n is missing in either of the above, 1 is assumed.

Note, if the numeric argument is missing from the :B, :F, or :X special characters, a value of 1 is assumed. Also, if the numeric argument of these special characters is an asterisk (*), the numeric value for the character is to be found in the argument list of the character I/O routine processing the format control list.

Examples:

(In the following examples, the symbol '[]' represents the final cursor position.)

- if format = 'HI, HOW ARE YOU?'

```
HI, HOW ARE YOU?[]
```

- if format = 'I AM FINE, THANK YOU.:N'

```
I AM FINE, THANK YOU.
[]
```

- if format = 'WATCH THIS! A:LB:LC:L:BD:RE'

```
WATCH THIS! A
             B
              C
E[]           D
```

- if format = 'TEST:TTABBING:TMECHANISM:N'

```
TEST____TABBING_MECHANISM.   (Note: '_' is strictly a
[]                           place holder for spaces,
                             i.e., a space character
                             really should appear
                             where '_' does)
```

- if format = 'SPACE:X5EXAMPLE:X3WITH:X3POSITIONING.:P26RE:N'

```
SPACE_____EXAMPLE___WITH REPOSITIONING.
[]
```

6.1.2 TRMPUT -

The calling sequence for TRMPUT is:

CALL TRMPUT (format-string, arg1, arg2, ...)

TRMPUT is a terminal output subprogram. It formats, converts, and prints its arguments to a user's terminal, under control of a format string. The format string will contain three types of objects: plain characters, which are simply copied to the terminal; special characters, which are used for cursor control; and conversion specifications, which are used for converting and printing arguments which follow the format string.

Each conversion specification found in the format string begins with the character '$'. Following the '$' there may be:

- an optional plus or minus sign which specifies right or left adjustment of the converted argument in the indicated field.

- an optional digit string representing a field repetition factor. If it is not present, it is assumed to be 1.

- a character which indicates the type of conversion to be applied.

- an optional digit string specifying a 'MINIMUM' field width (i.e., the field to be printed may actually be larger than specified, but it will not be smaller than specified); if the converted argument has fewer characters than the field width, it will be padded (on left or right, depending on the field conversion type and the field adjustment indicator) with blanks to make up the field width. If the character '*' is placed in this position, the next argument in the list (the one that follows the argument to be printed) is used to obtain the minimum field width.

- an optional period ('.') which serves to separate the field width from the next digit string.

- an optional digit string (the precision) which specifies the number of digits to be printed to the right of the decimal point of a single or double precision number, or the maximum number of characters to be printed from a character string. If the character '*' is placed in this position, the next argument in the list of arguments is used to obtain the maximum field width.

```
8)  N = TRMGET('$A20', S)

        N = 1; S = "abcd__efg_12jt"

9)  N = TRMGET('$A$C$C$C', S, T, U, V)

        N = 4; S = "abcd"; T = '_'; U = '_' V = 'e'

  (Note:  'C' conversion specifier suppresses space skipping)
```

The following example shows how the 'execution time' field width can be used.

```
10)  N = TRMGET('$A*$A*', S, 3, T, 5)

        N = 3; S = "abc"; T = "d__ef"
```

6.1.4 Some Notes On Terminal I/O -

Both TRMGET and TRMPUT are written in assembly language. TRMGET and TRMPUT are entry points into the routines FILGET and FILPUT, respectively (see Section 6.3). Essentially, the terminal I/O routines can be thought of as special cases of FILGET and FILPUT.

A call to TRMGET is equivalent to:

```
CALL FILGET (terminal LUN, format-string, arg1 ...)
```

A call to TRMPUT is equivalent to:

```
CALL FILPUT (terminal LUN, format-string, arg1 ...)
```

Note:

Each character is represented by a single element in an integer array. However, for convenience,the characters will be displayed here as contiguous character strings, addressed via the name of the array. Also, there is an EOS symbol attached to the end of all the character strings enclosed in double quotes. When the optional field width portion of the 'A' conversion specification is missing, the input field is a string of non-space characters. All initial space characters are skipped over and the input field is TERMINATED BY A SPACE CHARACTER or a NEW LINE. If the field width is specified, then any initial space characters are skipped over, and the FIELD WIDTH or a NEW LINE TERMINATES the input field.

```
 2)  N = TRMGET('$2A3', S, T)
         N = 2; S = "abc", t = "d__"
 3)  N = TRMGET('$/A $A5', S)
         N = 1; S = "efg_1"
 4)  N = TRMGET('$A $A $A', S, T, U)
         N = 3; S = "abcd"; T = "efg"; U = "12j5"
 5)  N = TRMGET('$/A3 $A1 $/2A3 $C', S, T)
         N = 2; S = "d"; T = '5' (no EOS; represented by single
                                  quotes)
 6)  N = TRMGET('$/2A $A', S)
         N = 1; S = "12j5"
 7)  N = TRMGET(':X2$A :P$A :P8 $A', S, T, U)
         N = 3; S = "cd"; T = "abcd"; U = "fg"
```

The conversion characters and their meanings are as follows:

I - The argument (1 word long) is converted to a decimal integer. If an adjustment indicator is not present in the conversion specification, the resulting output character string will be right adjusted within its field.

H - The argument is considered a half integer (1 byte long) Default field adjustment is 'right'.

L - The argument is considered a long integer (2 words long) Default field adjustment is 'right'.

O - The argument (1 word long) is printed out as an octal number. The resulting output string wil be right adjusted in its output field, when a justification indicator is not present.

A - The argument is taken to be a string of characters. The characters are stored in an integer array, one character per integer. Characters from the string are printed until an end of string character (0) is reached, or until the number of characters, indicated by the precision, is exhausted. If an adjustment indicator is not present in the conversion specification, the resulting output character string will be left justified in its output field.

S - The argument is taken to be a character string. The only difference between the 'S' and 'A' conversion specifiers is that the characters to be printed are stored in the FORTRAN hollerith data type, instead of the FORTRAN integer data type.

F - The argument is taken to be a single precision floating point number and is converted to the decimal notation of the form [-]mmm.nnnnnnn, where the length of the string of n's is defined by the 'precision' specification. The default 'precision' is 7. Default field adjustment is to the right.

E - The argument is taken to be a single precision floating point number and is converted to the decimal notation of the form [-]m.nnnnnnnE(+ or -)ee, where the length of the string of n's is defined by the 'precision' specification. The default 'precison' is 7. The default field adjustment is to the right.

D - The argument is taken to be a double precision floating point number and is converted to the decimal notation of the form [-]m.nnnnnnnnnnnnnnnD(+ or -)ee, where the length of the string of n's is defined by the 'precision' specification. The default precision is 16. Default field adjustment is to the right.

If no recognizable character appears after the '$', that character is printed; thus '$' may be printed by the usage of the string '$$'.

The following lines contain some example usages of conversion specifications and their resultant output (NOTE, '_' represents a space (' ') position).

if STR = 'abcde'	if INT = 578	if FLT = 352.709
$A = abcde	$I = 578	$F = 352.709
$-A = abcde	$-I = 578	$-F = 352.709
$+S = abcde	$+I = 578	$+F = 352.709
$A3.3 = abc	$I2 = 578 (*)	$F5.2 = 352.71
$S3.3 = abc	$-I2 = 578	$F5.2 = 352.71
$+A3.3 = abc	$+I3 = 578	$+F5.2 = 352.71
$S7 = abcde__	$-I5 = __578	$F9.4 = _352.7090
$-A7 = abcde__	$-I5 = 578__	$-F9.4 = 352.7090_
$+A7 = __abcde	$+I5 = __578	$+F9.4 = _352.7090

(*) An example of what happens when a numeric field width exceeds its "minimum" field width.

The following lines show examples of the execution time field width and precision portions of a conversion specification.

```
if STR = 'abcde', I = 3, and J = 7, then,
 CALL TRMPUT ('$A*', STR, I)
     'abcde' is printed at the terminal
 CALL TRMPUT ('$A.*', STR, I)
     'abc' is printed at the terminal
 CALL TRMPUT ('$A*.*', STR, J, J)
     'abcde__' is printed at the terminal
 CALL TRMPUT ('$A5.*', STR, J)
     'abcde' is printed at the terminal
```

TRMPUT signals its failure to write to the terminal by an approriate error message.

6.1.3 TRMGET -

The calling sequence for TRMGET is:

```
N = TRMGET (format-string, arg1, arg2, ...)
```

TRMGET is a terminal input function subprogram (It must be declared as a FORTRAN integer). It reads characters from the terminal, interprets them according to a format, and stores the result in its arguments. The format string usually contains specifications which are used to direct interpretation of input sequences.

A format string may contain:

- The special string characters representing blanks, tabs, newlines, backspaces, linefeeds, carriage returns, or formfeeds, which are ignored (these will be known as 'space' characters).

- Ordinary characters (not '$') which are expected to match the next non-space character of the input stream.

- Conversion specifications, consisting of the character '$', an optional assignment suppressing character '/', an optional space skipping suppressing character '-', an optional numerical field repetition factor, a conversion character, and an optional numerical 'MAXIMUM' field width specifier (i.e., the field to be read may actually be smaller than specified, but not larger than specified.)
  Note, the maximum field width specifier may be replaced by the character '*', which indicates that the argument following the one currently being processed contains the value of the maximum field width.

A conversion specification is used to direct the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppresion was indicated by the '/' character. The assignment suppression character '/' directs TRMGET to skip over the specified type of field in the input stream. An input field is defined as a string of non-space characters. However, an exception can occur when using an 'A' or 'S' conversion specification. See 'A' description.

The following conversion characters are permissable:

$ indicates that a single '$' character is expected in the input stream at this point; no assignment is done.

I indicates that a decimal integer is expected in the input stream; the corresponding argument should be of type integer.

H indicates that a decimal integer is expected in the input stream; the corresponding argument should be of type half integer (i.e., in DEC FORTRAN the type is LOGICAL*1 or BYTE).

L indicates that a decimal integer is expected in the input stream; the corresponding argument should be of type long integer (i.e., in DEC FORTRAN the type is INTEGER*4).

A indicates that a character string is expected in the input stream; the corresponding argument should be an integer array large enough to accept the string and an end-of-string (EOS = 0, the null character) symbol, which will be added. If an optional field width is not specified, the input record is terminated by either a space character or a newline.
If an optional field width is specified, only a newline may terminate the input record before the end of the field is reached. Thus, space characters may be embedded in the output field only if a field width is specified.

S indicates that a character string is expected; this specification is identical to the 'A' specification except that the corresponding argument should be a HOLLERITH field (i.e., in DEC FORTRAN this is the variable LOGICAL*1).

E
F indicates that a floating point number is expected in the input stream; the corresponding argument should be a single precision 'REAL' variable. The input format for a floating point number is a string of numbers, possibly containing a leading minus sign, followed by an optional exponent field containing an 'E' or 'D', followed by a possible signed integer.

D indicates that a floating point number is expected in the input stream; the corresponding argument should be a double precision 'REAL' variable. The input format is identical to that of the 'E', 'F' format.

C indicates that a single character is expected in the input stream; the corresponding argument should be an integer variable. Note, no EOS symbol is tacked on the end of the character in the output field.

TRMGET returns, as its value, the number of successfully matched and assigned input items. This can be used to decide how many input items were found.

If a program termination symbol (a null line or carriage return) is encountered by TRMGET, a -1 is returned.

If the OLPARS universal help symbol (?) is found as the first non-space character in the input record, a -2 is returned.

The following lines contain usage examples of the above conversion characters.

```
Current input string:  "579_34_62158" *
1)  N = TRMGET ('$I $II $2I1', J, K, L, M)
        N = 4, J = 5797; K = 3; L = 4; M = 6
2)  N = TRMGET('$2I2 $2I', J, K, L, M)
        N = 4; J = 57; K = 9; L = 34; M = 62158
3)  N = TRMGET('$/I $I', J)
        N = 1; J = 34
4)  N = TRMGET('$/2I$I', J)
        N = 1; J = 62158
5)  N = TRMGET('$/2I2 $I', J)
        N = 1; J = 34

Current input string:  "abcd__efg_12j5"
1)  N = TRMGET('$A $A3', S, T)
    N = 2; S = "abcd"; T = "efg" (see note on following page)
```

----
* '_' is strictly a place holder for spaces, i.e., a space character really should appear where '_' does.

Note:

Each character is represented by a single element in an integer array. However, for convenience,the characters will be displayed here as contiguous character strings, addressed via the name of the array. Also, there is an EOS symbol attached to the end of all the character strings enclosed in double quotes. When the optional field width portion of the 'A' conversion specification is missing, the input field is a string of non-space characters. All initial space characters are skipped over and the input field is TERMINATED BY A SPACE CHARACTER or a NEW LINE. If the field width is specified, then any initial space characters are skipped over, and the FIELD WIDTH or a NEW LINE TERMINATES the input field.

```
 2)  N = TRMGET('$2A3', S, T)
           N = 2; S = "abc", t = "d__"
 3)  N = TRMGET('$/A $A5', S)
           N = 1; S = "efg_1"
 4)  N = TRMGET('$A $A $A', S, T, U)
           N = 3; S = "abcd"; T = "efg"; U = "12j5"
 5)  N = TRMGET('$/A3 $A1 $/2A3 $C', S, T)
           N = 2; S = "d"; T = '5' (no EOS; represented by single
                                    quotes)
 6)  N = TRMGET('$/2A $A', S)
           N = 1; S = "12j5"
 7)  N = TRMGET(':X2$A :P$A :P8 $A', S, T, U)
           N = 3; S = "cd"; T = "abcd"; U = "fg"
```

8) N = TRMGET('$A20', S)

   N = 1; S = "abcd__efg_12jt"

9) N = TRMGET('ACCC', S, T, U, V)

   N = 4; S = "abcd"; T = '_'; U = '_' V = 'e'

(Note: 'C' conversion specifier suppresses space skipping)

The following example shows how the 'execution time' field width can be used.

10) N = TRMGET('$A*$A*', S, 3, T, 5)

   N = 3; S = "abc"; T = "d__ef"

6.1.4 Some Notes On Terminal I/O -

Both TRMGET and TRMPUT are written in assembly language. TRMGET and TRMPUT are entry points into the routines FILGET and FILPUT, respectively (see Section 6.3). Essentially, the terminal I/O routines can be thought of as special cases of FILGET and FILPUT.

A call to TRMGET is equivalent to:

```
CALL FILGET (terminal LUN, format-string, arg1 ...)
```

A call to TRMPUT is equivalent to:

```
CALL FILPUT (terminal LUN, format-string, arg1 ...)
```

6.2 CLPARS TERMINAL GRAPHICS INPUT/OUTPUT

Along with the terminal input/output routines, CLPARS has a set of graphics input and output routines. Most of the output routines are utilities closely resembling the PLTMAP FORTRAN routines requested by the buyer. The graphics input routine, necessary for some CLPARS functions, is not in the PLTMAP routines, and had to be written. The following sections discuss in more detail the actual utilities chosen to be part of the CLPARS graphics I/O.

6.2.1 Graphics Input Utility -

The graphics input routine (GIN) is used to obtain graphic display screen coordinates. First, this routine places the graphics terminal into graphics input mode. It then waits for the return of a graphics screen coordinate and the character that was typed in to send the coordinate. If the display terminal does not have cursor wheels, a joystick, or some other hardware graphics cursor manipulator, GIN must take over this function (i.e., special characters will have to be reserved for graphics cursor movement; possibly l-left, r-right, u-up, d-down). This routine is a system-dependent program.

GIN's program usage would be:

```
CALL GIN(X, Y, CHAR)
```

where all of GIN's arguements are of integer type. X and Y are the terminal horizontal and vertical coordinates, respectively. CHAR

will be the character used to send the coordinates back to the calling program.

6.2.2 Graphics Output Utilities -

The following depicts the type of graphics output functions that CLPARS needs to have for its display purposes:

- the ability to place a text string at a specified point
- the ability to draw lines
- the ability to erase the screen and "home" (move to the upper left corner) the cursor.

The subsequent paragraphs describe the routines that were chosen from the PLTMAP utilities for the OLPARS graphics display, plus two additional routines written, MOVE and RCTNGL.

6.2.3 TEXT -

This routine places a string of text on the graphics display, starting at a given set of coordinates. The screen coordinates are relative screen coordinates used to redefine the origin. Each character of the string resides in a separate integer, and the last integer position in the string must contain zero.

6.2.4 MARK -

This routine places a given marker at a specified set of screen coordinates. The markers are those specified in the PLTMAP utilities.

CLPARS uses this in its two-space display programs.

6.2.5 LINSEG -

This routine draws a line between two specified points. It is used to draw display borders and data and logic tree structures.

6.2.6 ERASE -

This routine erases the terminal screen (and "homes" the cursor, if not done automatically).

6.2.7 MOVE -

This routine moves the terminal screen cursor to specified terminal coordinates.

6.2.8 RCTNGL -

This routine draws the outline of a rectangle given the coordinates of two opposite corners.

6.3 CLPARS TEXT FILE INPUT/OUTPUT

Any CLPARS text file input routine will need a way to read numeric characters from a file and convert then to binary information. Likewise, a file output routine needs a way to convert binary data into printable characters. The following two sections describe the subroutines that will enable the CLPARS file I/O routines to operate.

6.3.1 FILGET -

The calling sequence for FILGET is:

N = FILGET (lun, format-string, arg 1, arg 2, ...)

FILGET is a file input function subprogram. It reads characters from a file, specified by a logical unit number (lun), interprets them according to a format (identical to TRMGET format), and stores the results in its arguments. The format string usually contains specifications which are used to direct interpretation of input sequences. See Section 6.1.3. for further information on the format string.

FILGET returns as its value the number of successfully matched and assigned input items. When the end of the file is reached, FILGET will return a -1 as its function value. (On RSX-11M, if the file being read is actually a terminal, a CTRL-Z will simulate the end of file.) When the terminal is being read by FILGET, a program termination symbol (null line) will also cause a -1 to return.

This is not true when FILGET is reading a file.

6.3.2 FILPUT -

The calling sequence for FILPUT is:

CALL FILPUT (lun, format-string, arg 1, arg 2, ...)

FILPUT is a file output subprogram. It formats, converts, and prints its arguments to a file under control of a format string (see Section 6.1.2.). The file is specified by a logical unit number (LUN).

FILPUT signals its success or failure at writing to the file with an appropriate error message. Both FILGET and FILPUT are assembly I/O routines.

6.3.3 Printing - CLPARS Output To A Computer Printer -

There are a few CLPARS programs that give the user the ability to obtain printable listings (or text files) of a variety of CLPARS data forms. Cluster plots, rank order matrices, and confusion matrices are a few examples of the CLPARS data forms available for printing.

The print utility commands create a file within the system, to store the output to be generated. Once the printer listing generation is complete, the utility makes a call to a system dependent program (PRINTR) that will send the file to a printer for printing.

The following is a list of print utility commands that may be implemented. (The starred commands have not yet been implemented).

PRTRNK* - used for rank order displays or probability of confusion or discriminant measure.

PRTDS - used to obtain certain basic statistical information about a data base.

PRTCM - used to print out a confusion matrix.

PRTCLP* - used to print out a cluster plot.

PRTIDX - used to index a particular display or set of display symbols, and obtain information about the vector that the display symbol represents.

PRTLOG - used to list the logic of the current logic tree.

PRTOSD* - used to produce a printer simulation of a one-space display (micro view format only).

6.3.4 CLPARS Data Tree Input/Output -

Users of CLPARS will probably have their data on various mediums, such as disk or magnetic tape. CLPARS must be able to read and convert a user text file, containing data, into an CLPARS tree (that is, create a tree information and tree vector file with the users' data).

Also, a user may want to take an existing CLPARS data tree and create a text file of vectors to edit. Therefore, CLPARS must convert its data tree to a "system" text file.

To make the programming of the above functions as minimal as possible, all input and output data will have the same CLPARS logical format. This format is specified in the programs FILEIN and FILEOUT (for the RSX-11M system). Note, the output from FILEOUT can be used as input to FILEIN.

6.3.5 Some Notes On Terminal And Text File I/O -

The internal binary representation of character strings within a computer program varies from program language to program language, from operating system to operating system, and from computer to computer. The terminal and file I/O routines (TRMGET, TRMPUT, FILGET, FILPUT) can alleviate this problem of internal character representation by using an integer index (into a table of characters) which points to the character to be represented. This would make CLPARS I/O character strings system independent.

Under RSX-11M we have chosen not to include such a table within the terminal and file I/O routines, because the ASCII 7 bit character set and its internal form within Digital Equipment Corporation FORTRAN already give us a convenient integer representation for character strings.

Even though the terminal I/O routines appears to be a special case of file I/O routines, there are some differences. For instance, the terminal I/O routines will automatically "open" the terminal logical unit number (if it is not open) while the file I/O routines do not.

SECTION 7

OTHER FEATURES

7.0 CLPARS FORTRAN CODE GENERATION

A user may create FORTRAN code, which becomes a subroutine called by other programs. This method is used in the measurement transformation command (MEASXFRM). Note, the FORTRAN code generation program (in this case, MEASXFRM) may or may not be able to call up a FORTRAN compiler to compile the FORTRAN subroutine. On systems that do not allow initiation of a FORTRAN compiler from within a program, the user will have to compile his/her CLPARS generated FORTRAN program. Under RSX-11M, CLPARS will be able to call the FORTRAN compiler (via a command file).

Example of user generated FORTRAN code:

Function Description: MEASXFRM is a means of transforming one data set with dimensionality M into another data set of dimensionality N (N may or may not = M). This transformation is done by means of character arithmetic expressions. Measurement 'i' in the new tree is symbolized by NM(i). Measurement 'i' in the old tree is symbolized by OM(i).

For example, suppose we have, as the current data set, a tree with dimensionality four and wish to create another tree with dimensionality five. Furthermore, suppose each measurement in the new tree is to be the same as that in the old tree, with the exception that measurement five of the new tree is to equal the sum of measurements three and four of the old tree.
The user would enter the following FORTRAN statements:

```
NM(1) = OM(1)
NM(2) = OM(2)
NM(3) = OM(3)
NM(4) = OM(4)
NM(5) = OM(3) + OM(4)
```

In the RSX-11M version of portable CLPARS, MEASXFRM is implemented via a command file. An editor is initiated to accept the above FORTRAN statements. The statements are inserted into a subroutine which is compiled and linked with the rest of the MEASXFRM program. The command file then causes the program to execute. For more details, see MEASXFRM in the CLPARS VI Program Specifications.

An additional capability to save the FORTRAN subroutine is available to the user. This way, (s)he can easily use the measurement transformation created under MEASXFRM many times.

7.1 OLPARS BOOLEAN STATEMENT INTERPRETER

There are several other instances where an OLPARS user may enter linguistic or Boolean statements in the course of analysis (e.g., linguistic partition in structure analysis, linguistic group logic or linguistic reject regions in logic design). In each of the cases, the user enters exactly one statement and vectors are tested against this statment to see if a true or false value is obtained. In order to make these programs (LINGPART, LINGLOG and LINGRJCT) system independent, an interpreter program (INTRPB; Interpret Boolean) is incorporated into OLPARS to perform the above test. This program, which will be written in FORTRAN, will be system independent.

The Boolean statement may consist of the following information:

1. Measurement positions in a data vector.
2. Floating point numbers
3. Logical Operators
4. Arithmetic Operators
5. Arithmetic Functions

Additional advantages of this scheme are:

- o No command file need be created.
- o User interaction is simplified.
- o The interpreter can check the syntax of the Boolean statement and it returns with an error flag if the syntax is incorrect.

7.2 BATCH PROCESSING IN OLPARS

Batch processing in CLPARS is a system dependent operation. For instance, MULTICS, a multi-user time sharing computer system, allows "batch" processing (known as absentee jobs) to occur via a command file (see Figure 7-1). The programs to be run, along with the input to these programs, are all contained in a special command file. The output from the programs, directed to a user terminal, is placed in a special file within the user's directory for examination at a later date. The absentee job is treated as if there was a user at some terminal.

It is obvious that considerable familiarity with the interactive queries of MOOS (MULTICS OLPARS Operating System) functions is necessary since all queries must be correctly answered with the command file. The absentee job in Figure 7-1 could be performed on a number of data sets by simply changing the tree name "datatree" to the names of other data sets.

Under the RSX-11M system, the "AT" processor handles the command file (batch) processing. In this system, the program's terminal input does not come from a file, and the program's terminal output is not put into a file. This means that the "AT" processor does not treat a command file as a separate terminal. (Hence, it will not be used in the RSX-11M portable CLPARS).

To make this "batch" processing transportable to different machines, there should be two separate CLPARS. The difference between the two systems will occur in the terminal I/O packages; one package will communicate directly with the terminal, the other (in the batch system) will communicate with files (i.e., batch OLPARS will get its commands from one file and put its terminal output into another file).

The total size of the batch system could also be reduced by keeping only the programs that create character output in the system. This decision will be left up to the people who want a batch version of OLPARS on their computers.

An example of an CLPARS command (batch) file can be found in Figure 7-2.

cwd > udd > C > OLPARS	(change working directory to OLPARS)
restore data tree	(restore the design data set)
4	(answer questions related to
300	console type and baud rate)
fisher	(invoke fisher pairwise logic)
0	(select option 0)
1	(select 1 threshold)
8	(set minimum vote count to 8)
yes and	(hard copy confusion matrix
yes printer)	list of errors to line
no	(halt fisher calculation)
logout	

Figure 7-1 A MULTICS Command File

SETDS	(set data set)
/	
TREE1	(answer questions related to treename and nodename
NODE1	(for the current data set desired)
/	
FISHER	(invoke fisher pairwise logic)
/	
0	(select option 0)
1	(select threshold)
4	(set minimum vote count to 4)
YES	(send confusion matrix and a list
YES	of errors to line printer)
NO	(halt fisher calculation)
/	

NOTE: "/" delimits answers to program questions

Figure 7-2 Example of CLPARS Command File

Within an RSX-11M OLPARS, the batch mode will be handled in the following manner. A separate "batch" OLPARS would be in existance. A user would create the command file via an editor program. (S)He then runs the batch OLPARS (BOLP) while in his/her OLPARS directory. BOLP executes the command contained within the command file. Once BOLP is started up, the user could not run the interactive version of OLPARS, but could run other tasks within the RSX-11M system. Located within the user's directory, would be the file containing all of the terminal output from the batch OLPARS session. The user could use a system command (PIP for instance) to view the contents of the file after execution of BOLP.

7.3 EXPANDABILITY

Adding new applications programs to FORTRAN OLPARS, or inserting additional options into already existing programs, should be a relatively simple task for the following reasons.

- o If a new command is to be added, the entire system does not need to be recompiled or have its tasks rebuilt. The new program must be compiled together with the OLPARS subroutines it needs, and its name must be inserted in the list of allowable command names that is accessed by the CIP (or the operating system).

- The modular construction of the CLPARS programs will mean that a good part of the code for any new program will already exist.

- New options in existing routines can be programmed by changing only the particular routine and, most probably, by changing only a few of the subroutines of that particular routine. Many of the CLPARS commands have been designed to accommodate additional functions by adding subroutines and giving the user additional options.

SECTION 8

SYSTEM DEPENDENCIES

In a system as complicated as OLPARS, where there is a large amount of input/ output with files, the printer, and especially the terminal, there must be system dependencies to handle these features as well as others. In order to make OLPARS as portable as possible, the system dependencies have been isolated into a set of routines that will be called by the rest of OLPARS to perform the dependent functions like I/O. In this section, we review the system dependent functions of portable OLPARS with explanations here, or with references to documentation elsewhere.

o Start Up and Sign Off

Signing on and off OLPARS is system dependent. OLPARS needs an initialization routine to set terminal characteristics in files, or initialize other files. When there is no executive program, the user must call this routine (HELOLP) when (s)he first starts up the system. If an executive command input processor (CIP) program exists, then this initialization routine can be called automatically for the user.

When the operating system has special interrupt functions to abort programs, signing off of OLPARS can be slightly tricky. If OLPARS programs do not have the ability to catch the abort interrupt signal, it could very well mess up the OLPARS file

structure. If this is the case, users should avoid the signal abort feature on their system when running CLPARS. Under RSX-11M FORTRAN, a user may type a control-Z (^Z) when entering input to a program. This simulates an end-of-file interrupt signal. Since this interrupt can be caught by programs, RSX-11M OLPARS users need not worry about using it.

OLPARS programs should leave gracefully, so there is a standard procedure for exiting OLPARS programs. If a user enters a null line during an input session with an "OLPARS program", the program will exit. To exit "CLPARS" (that is, the CLPARS comand level) the user must type BYEOLP; a null line will not allow system termination at the command level.

- o *The Command Input Processor (CIP) (See Section 2 and Appendix A)*

- o Spawning (See Appendix A)

- o Compilation and Execution of FORTRAN Statements typed in by the user (see Section 7.0)

- o Terminal and File Text I/O (see Section 6)

- o Use of a "Batch" OLPARS (see Section 7.1)

- o Programmer Aid Options (see Appendix G)

- o Character Handling (see Section 6)
    - oo character string comparisons
    - oo character string terminator
    - oo character string length determination

o Processing bit maps in the LI file (see Section 4.6)

o Creating "system" file names from CLPARS tree names

On computer operating systems that have file names which are smaller than CLPARS tree names, a unique, algorithmically generated file name must be produced from the tree name. An alternative method would be to create a set of unique file names, having nothing to do with an CLPARS tree name. The file name would be associated with the tree name through some file or table. (See discussion of File Code Table in Appendix E).

APPENDIX A

THE COMMAND INPUT PROCESSOR (CIP)

or

How to "talk" to CLPARS on RSX11M version 3.2

This section describes the programs and files necessary for contriving the Command Input Processor (CIP), "login" function (HELCLP), and "logout" function (BYECLP) under the DEC operating system RSX11M version 3.2 (reader must be familiar with this operating system).

The CIP program only partially implements the CIP function. Its job is to prompt the user for an CLPARS command name and make sure what the user has typed is valid (it accepts initial substrings of all commands, except BYECLP). It produces a command file which invokes the OLPARS command requested by the user (explained later).

The CMINIT program, which initializes the terminal screen parameters and CLPARS system directory string* in the user's Communications (CM) file, partially implements the HELCLP function (hello OLPARS, or "logging in"). It welcomes the user and displays

---

* The operating system manager must "install" the two programs CIP and CMINIT as "...CIP" and "...CMI", respectively, because they both need to access their system command line (via system supplied subroutine, GETMCR) for the CLPARS system directory location.

a list of terminal types known to CLPARS. It asks the user to name the terminal type (s)he is using. The program uses the name typed in to find a file containing the screen parameters to be placed in the user's Communications file. It is important that the correct screen parameters are obtained for the given type of terminal; otherwise, erroneous results may occur when using "interactive" CLPARS displays. CLPARS essentially remembers the last terminal at which the user has "logged in", so supplying an empty response to the CMINIT prompt is perfectly safe as long as the operator is at the same type of terminal (s)he has used at his/her last CLPARS session.

Three command files complete the implementation of the CIP, HELOLP, BYECLP functions. The files are "HELOLP.CMD", "CLPDIR.CMD", and "BYECLP.CMD". The user "logs" into CLPARS by invoking the HELOLP command file*. The command file locates the CLPARS directory by invoking the "CLPDIR.CMD" file**. HELOLP then queries the user for the "login" name assigned to him/her by the CLPARS manager. It uses the "login" name to find out where the user's CLPARS directory is located. Next, the CMINIT program is

---

* The user may type "@HELOLP" or "@[directory-specification]HELOLP" to start up the command file. A "login" request will follow. To skip the "login" request, the user may type "@HELOLP <user name>", where <user name> is the user's "login" name.

** "CLPDIR.CMD" must be located in the same directory in which "HELOLP.CMD" resides. "CLPDIR.CMD" should be modified by the person responsible for maintaining CLPARS so that it points to the directory in which all the CLPARS commands reside (the CLPARS system directory)

invoked by HELCLP. Finally the CIP program is activated and CLPARS is ready to do work for the user.

The command file created by the CIP program (called "CLPCOM.CMD") is invoked by the HELOLP command file after the CIP program terminates. The "CLPCOM.CMD" file contains the necessary information needed to start up the user requested CLPARS command. It also lets the HELOLP command file "know" which CLPARS command has been executed. If the CLPARS "logout" command (BYECLP) was requested by the user, the "BYECLP.CMD" file is invoked. Once "BYECLP.CMD" is finished, the HELOLP command file stops its own execution.

The following two pages illustrate the OLPARS Command Input Processor, "login", and "logout" functions, and relationships between RSX command files and OLPARS programs implementing these functions.

```
HELOLP COMMAND FILE (HELOLP.CMD)
---------------------------------
| 1. invoke OLPDIR.CMD            |
| 2. query user for login name    |
| 3. call CMINIT program          |
|                                 |
|    repeat                       |
| 4.   call CIP program           |
| 5.   invoke OLPCOM.CMD          |
|    until (user types "BYEOLP")  |
---------------------------------

OLPDIR COMMAND FILE (OLPDIR.CMD)
---------------------------------
| locate OLPARS system directory  |
---------------------------------

CMINIT program
---------------------------------
| 1. ask user for terminal type   |
| 2. initialize terminal screen   |
|    parameters and OLPARS        |
|    directory string in user     |
|    CM file.                     |
---------------------------------

COMMAND INPUT PROCESSOR program
---------------------------------
|    repeat                       |
| 1.    prompt user for OLPARS    |
|       command                   |
|    until (command is valid)     |
|                                 |
| 2. create OLPCOM.CMD            |
---------------------------------

OLPCOM COMMAND FILE (OLPCOM.CMD)
---------------------------------
| initiate user requested OLPARS  |
| command                         |
---------------------------------

BYEOLP COMMAND FILE (BYEOLP.CMD)
---------------------------------
| clean up loose ends             |
---------------------------------
```

OLPARS Command Input Processor, "login", and "logout" functions Summary

AD-A118 731 PAR TECHNOLOGY CORP NEW HARTFORD NY F/G 9/2
ON-LINE PATTERN ANALYSIS AND RECOGNITION SYSTEM. OLPARS VI. PRO--ETC(U)
JUN 82 S E HAEHN, D MORRIS
UNCLASSIFIED PAR-82-15 NL

3 OF 3
AD A
118731

END
DATE
FILMED
09-82
DTIC

```
                     ===========================
                    |  Any RSX11M directory      |
                    |                            |  ** contains
                    |   -----------------------  |     strings
                    |  | HELOLP.CMD            | |        |
 ===============    |  |                       | |      IMLAC
| OLPARS system |   |  |///////////////////////| |      TEK4014
| directory     |<---------    OLPDIR.CMD      | |      TEK4051
|               |   |  |///////////////////////| |
|               |   |  |                       | |
|               |   |  |  Login Name Validation| |   ===============
|               |   |  |  and User Directory   | |  | OLPARS user   |
|               |   |  |  Information   ------------>| directory     |
|  -------------|   |  |                       | |  |               |
| |TERMINALS.TXT-----------                      | |  |               |
| | **          |   |  |   \                   | |  |               |
| |-------------|   |  |    \                  | |  |  ------------ |
| |TEK4051.TXT  -------------> CMINIT --------------->| CM.CLP     | |
| |-------------|   |  |    /                  | |  |  ------------ |
| |IMLAC.TXT    ------------/                  | |  |               |
| |-------------|   |  |  /                    | |  |               |
| |TEK4014.TXT  ----------                       | |  |               |
|  -------------|   |  |                       | |  |               |
|               |   |  |                       | |  |               |
|  -------------|   |  |                       | |  |  ------------ |
| |PROGRAMS.TXT ------------->    CIP    ------------>| OLPCOM.CMD | |
|  -------------|   |  |                       | |  |  ------------ |
|               |   |  |                       | |  |        |      |
|               |   |   -----------------------  |  |        |      |
|               |    ===========================    |        |      |
|  -------------|                                   |        |      |
| |BYECLP.CMD   |                                   |        |      |
|  -------------|                                   |        |      |
|       .       |<-------------------------------------------       |
|       .       |                                   |               |
|Olpars Commands|                                   |               |
|               |                                   |               |
 ===============                                     ===============
```

Relationships between RSX command files and
OLPARS programs implementing the Command
Input Processor (CIP), "login", and "logout"
functions

APPENDIX B

OLPARS RSX-11M SYSTEM DEPENDENT FILES

B.1 The OLPARS Directory

Each OLPARS program including the CIP, resides as a segment task (.TSK) file in the OLPARS directory. There are also several purpose files which reside in this directory. These include the OLPARS option file (OPTION.OLP) the OLPARS option text file (OPTION.TXT), the OLPARS terminal screen coordinate files (including TERMINALS.TXT), and the OLPARS program dictionary (PROGRAMS.TXT). These files are described in the following sections.

B.1.1 OLPARS Option File

The option file contains the names of all OLPARS commands, an associated option number, and a possible list of other options (programs) that are associated with a given option. Under RSX-11M, the option file has the name "OPTION.OLP" and is a fixed length record (block I/O) file.

The first word of the first record of the option file contains the number of options in the file. The next portion of the file contains the option name - option number associations. This part of the file has the option names in alphabetical order. The next

section holds the option numbers and option list pointers. The option numbers are sorted into ascending numerical order. The last section contains the option list, which is nothing more than a list of pointers to the option names in the option name-option number section. The first number in the option list is actually the number of options contained in the list. (See Figure B-1).

The two CLPARS programs that access this file are SETOPT and GETOPT. SETOPT looks through the option name-option number section for an option name. It returns to its calling program the option number of the option name for which it was looking. If SETOPT cannot find the given option name, it will return 'ANYTHING's option number. GETOPT scans through the option number-option list pointer section for an option number. It will return to its calling program the names of the options pointed to by the elements of the option list section. Note that an option may have more than one option list associated with it, i.e., multiple menus (see Figure B-1, option #20). It is also possible that an option does not have an option list (e.g., option #90 in Figure B-1).

B.1.2 OLPARS Option Text File

A system dependent program called MAKOPT creates the options file (OPTION.OLP) from a source text file (OPTION.TXT) containing each OLPARS command with its option number and option list. This program should be used only by OLPARS programmers/maintainers to create the proper option file (see Appendix G for usage of MAKOPT).

OPTION FILE RECORD NO.						
1	9	# of options				
2	ABCD	10				
3	AKLMD	20				
4	DEFG	40				
5	EFG	100				Option name - Option number
6	FDX	50				Section
7	LVD	60				(header)
8	STU	70				
9	VDA	80				
10	ZBD	90				
11	10	20				
12	20	21	22	23		
13	40	24				
14	50	25				Option number - Option list pointer
15	60	26				Section
16	70	27				(logical entry)
17	80	28				
18	90	-1				
19	100	29				
20	4	4	5	3	8	
21	2	7	4			
22	2	8	9			
23	1	6				Option list
24	1	8				Section
25	1	5				
26	3	8	4	6		(logical entry)
27	2	4	2			
28	2	3	5			
29	1	8				

Figure B-1 OLPARS Option File (OPTION.OLP)

The format of the source text is as follows:

PROGRAM NAME : OPTION # : MENU # : OPTION 1 ... OPTION 15

The program name refers to an OLPARS command. The maximum length of a program name allowed in the OPTION.CLP file is 10 characters. The option number is an arbitrarily assigned unique number.

Once the option number is assigned to a program, that program should always retain the same option number. The reason for this is that user files (logic trees, for instance) retain the option number (of the program that created them) as data. If the option number of an existing OLPARS program were changed, all the user files containing that option number would now have incorrect option numbers.

The menu number represents the different possible menus (option lists) that a single OLPARS command might display. The menu numbers need not be in order, but they must be continuous (that is, if there are four different menus, the menu numbers used must be 1, 2, 3 and 4).

The option list is a set of "other" OLPARS commands that may be used after "PROGRAM NAME" has completed it computations. The command names are separated by spaces. The last option should have a "slash"(/) following it. The slash must be separated from the last option by at least one space (see Figure B-2).

```
;     This example format
;     was used when producing
;     the option file found
;      in Figure B-1.
;
;
ABCD   :  10  :  1  :  DEFG EFG AKLMD STU /
AKLMD  :  20  :  1  :  LUD DEFG /
AKLMD  :  20  :  2  :  STU VDA /
AKLMD  :  20  :  3  :  FDX /
DEFG   :  40  :  1  :  STU /
EFG    : 100  :  1  :  STU /
FDX    :  50  :  1  :  EFG /
LVD    :  60  :  1  :  STU DEFG FDX /
STU    :  70  :  1  :  DEFG ABCD /
VDA    :  80  :  1  :  AKLMD EFG /
ZBD    :  90  :  1  :   /
```

Figure B-2 OLPARS Option Source Text File

The maximum number of options allowed in the option list is 15.

A semicolon found in column one of any text record will denote a comment line (a line to be ignored by the MAKOPT program). Blank lines are not allowed in the file (they cause MAKOPT to quit reading the file). The option list may extend over several lines. If this occurs, comments MUST NOT separate the lines.

In the example given (Figure B-1), the programs names are in alphabetical order. This is necessary to create a proper option list file.

In the current 'OPTION.TXT' file a few of the program names have been commented out because of space considerations within the MAKOPT program. Those that have been commented out will name 'ANYTHING' as their option list (see SETOPT in section B2). In fact, any program (with a name in the 'OPTION.TXT' file) that does not alter the mathematical projection of a data set, or the actual vectors within the current data set may be commented out.

B.1.2.1 Future Option File Maintenance

A modification to the current option file scheme would give CLPARS maintenance people better control over the consistency of the "ANYTHING" command found in CLPARS. A description of how the "ANYTHING" command works is now in order.

Currently, ANYTHING is a very simple program. It simply reads a text file (ANYTHING.TXT) and prints it out to the user's terminal. The file contains a list of all the CLPARS programs and the category in which they reside. This means that whenever a new command is added to CLPARS, both the OPTION.TXT file and ANYTHING.TXT file must be modified. This could cause consistency problems for an CLPARS maintenance person if (s)he forgot (heaven forbid, not me) to alter both files. To subvert this problem, changes would have to be made to the OPTION.TXT file (thus, the OPTION.CLP file), the MAKOPT program and the ANYTHING program.

1. To the OPTION.TXT file, a category code would be added to each program record. Note, all commands would have to be entered into the OPTION.TXT file (not "commented" out like some are now to save array space in MAKOPT).

2. MAKOPT would have to be changed to handle the category code. To save space in the program, a temporary file would have to be created for option list storage (which means a "re-think" on the whole MAKOPT algorithm).

3. The ANYTHING program would be re-written to read the new OPTION.CLP file created by MAKOPT (The ANYTHING.TXT file is no longer used). It would then organize the program names according to their category code, before printing out the results to the terminal.

For an alternative to the above, MAKOPT could do all the categorization work and add another list to the OPTION.CLP file with all the programs placed within their own categories. Then ANYTHING would be less complicated than previously mentioned. This modification (on system dependent programs) is only suggested for easier maintenance of CLPARS.

B.1.3 CLPARS Terminal Screen Coordinate Files

The terminal screen parameters (see section 5.1.4) are stored in individual text files found in the CLPAR system directory. The name of these files are stored in the file TERMINALS.TXT, also found in the same directory. When a user "logs" into CLPARS (see Appendix A), the TERMINALS.TXT file is displayed at the user's terminal. They, in turn, choose one of the terminal names (file names, without the filename type or extension name added) from the list displayed. The file type, ".TXT", is attached to the terminal name chosen. The resultant name is used to access the screen parameters file. The screen parameters are stored in the user's Communications file. The following text shows an example of the contents of a TERMINALS.TXT file and the contents of a screen parameters file.

Example Contents of TERMINALS.TXT

```
tek4014a - Tektronix 4014 with smallest size characters
tek4014b - Tektronix 4014 with next to smallest characters
tek4014c - Tektronix 4014 with next to largest characters
tek4051  - Tektronix 4051
IMLAC    - Tektronix compatability mode only
```

Example Contents of TEK4051.TXT

```
640      Ws  - no. of display units in screen width
480      Hs  - no. of display units in screen height
460      Wd  - no. of display units in display width
391      Hd  - no. of display units in display height
  9      Wc  - no. of display units in char. width
 14      Hc  - no. of display units in char. height
 88      a   - x coord. of lwr. left corner of display
 75      b   - y coord. of lwr. left corner of display
548      c   - x coord. of upr. rt. corner of display
466      d   - y coord. of upr. rt. corner of display
 28      MrW - number of rows in a cluster plot grid
 40      Ncl - number of cols. in a cluster plot grid
 74      LcH - number of characters in a screen line
  0      Not used, yet
133      e   - x coord. of left side of 1 space base line
 96      f   - y coord. of lowest 1 space base line
503      g   - x coord. of right side of 1 space base line
 10      Nc  - max. no. of classes on 1 sp. macro plot
 28      Ds  - distance between macro base lines
```

B.1.4 OLPARS Program Dictionary

The OLPARS program dictionary (PROGRAMS.TXT) is a simple text listing of all the OLPARS commands, kept in alphabetical order. The Command Input Processor (CIP) uses this list to verify whether or not the OLPARS user has entered a legitimate OLPARS command. The file may contain same line comments. There must be at least one space between the program name and the comment.

Example of a Program Dictionary:

```
ANYTHING ; This programs displays the ANYTHING.TXT file
BYECLP   ; program used to exit CLPARS
CRANDTS  ; create a random data set
DDATANOD ; delete a node from a data tree
    .
    .
    .
```

B.2 The User's Directory

Each CLPARS user must have an RSX-11M directory to store local CLPARS user files. These include fixed and variable files, temporary sequential files containing information to be sent to a line printer, and ASCII data files. The fixed and variable files have been discussed in great detail in earlier sections. The sequential files for line printer output are created and opened by a system dependent program OPENS. They are written into by the routines that produce line printer output. An operating system utility (PIP, on RSX-11M) will cause the file to be printed and then deleted (depending on local operating system configuration) from the user's directory.

The ASCII data files are used during the process of reading or writing data to files. In order to create an CLPARS data tree from data within a file in the user's directory, the data must be in CLPARS data file input format (see the description for FILEIN in the CLPARS User's Manual). Similarly, the data file created by FILEOUT (the CLPARS data set dumping routine) is put into the OLPARS data output format (see FILEOUT in the CLPARS User's Manual). The file created by FILEOUT can be used as input to FILEIN. Two of the user files which are system dependent are the File Code Table (FCT.CLP) and History (HS.CLP) file. The FCT is described in the next section. The History file is described in Appendix E, along with the CLPARS instrumentation package.

E.2.1 File Code Table

The CLPARS File Code Table (FCT) contains the system names of the fixed and variable files that belong to an CLPARS user. The system names of the fixed files reside in the first 10 entries of the FCT as follows:

File	Entry (File Code)	RSX-11M File Name
Communications	1	CM.CLP
Tree List	2	TL.CLP
Logic List	3	LL.CLP
Display Information	4	DI.CLP
Display Value	5	DV.CLP
Projection Vector	6	PV.CLP
Scratch 1	7	S1.CLP
Saved Vector	8	SV.CLP
Saved Transformation Matrix	9	SM.CLP
History	10	HS.CLP

Under RSX-11M, the FCT will be a direct access file (using CLPARS "block" I/O) with a record size of 8 words. The first record, the header, will contain a pointer to the next available entry to be filled (first entry in the free list link), the number of filled entries, a pointer to the last entry in the free list link, and a pointer to the last entry in the table. These four integers will be found in the first four words of the first record (see Figure E-3).

header	next available entry = 16 number of filled entries = 14 last entry in free list link = 19 last entry in file code table = 19	
entry 1	0	CM.CLP
2	0	TL.CLP
3	0	LL.CLP
4	0	DI.CLP
5	0	DV.CLP
6	0	PV.CLP
7	0	S1.CLP
8	0	SV.CLP
9	0	SM.CLP
10	0	HS.CLP
11	0	LOGIC231.OLI
12	0	LOGIC231.OLV
13	18	TWENTYON.OLV
14	13	TWENTYON.OLI
15	0	ALBERTAS.OTI
16	14	
17	0	ALBERTAS.OTV
18	19	
19	-1	

Figure B-3 File Code Table (file)

The rest of the file contains the RSX-11M file name records that CLPARS needs to access the user's fixed and variable files. A file name record consists of a free list link pointer and a file name. In the first word of the file name record resides the free list link pointer. The last seven words contain the file name, including a null character (zero in CLPARS). When the free list pointer is zero, the file name portion of the record contains the file name of an existing RSX-11M file. When the pointer is non-zero, the file name record resides in the free list of file name records, i.e., the file name record is available for use by OLPARS. In Figure B-3 entries 13 and 14 are in the free list even though they have an old file name left in the file name portion of the record.

The "free list" is a linked chain of file name records. The head of the chain is found in the first record of the FCT. It is called the next available entry pointer (see above).

The last link in the free list chain contains a -1, an invalid link value. If the File Code Table is full, both the next available entry pointer and the last entry in the free list link equal zero. If the next available entry and the last entry in the free list link are equal, but not zero, then there is one free entry left in the File Code Table. This entry's link will contain the -1 mentioned above.

The size of the File Code Table will determine the number of trees an CLPARS user may have within his CLPARS directory. The size of a user's FCT can be determined by an CLPARS installation manager when (s)he creates a user's directory. It may also be extended at a later date.

APPENDIX C

STEPS TO TAKE IN EXPANDING CLPARS UNDER RSX-11M

o Adding a command to CLPARS:

1. Put the name of the command in the program dictionary (can use EDI editor utility) in alphabetical order. The program directory is in the CLPARS directory.

2. Add the appropriate entries to the OPTION.TXT file (using EDI) in the CLPARS directory and execute the program MAKOPT for a new options file.

3. Create a Help file for the command within the help directory and add the command name to the help dictionary(see Appendix D).

o Adding a user to CLPARS:

1. Create the user's "system" directory with the UFD utility (provided by DEC on RSX-11M).

2. Add the user's "login" identification along with the User Identification Code (UIC) of his/her newly created directory to the HELOLP.CMD command file, found in the

CLPARS directory (use EDI).

3. Use the GEN programmer's aid to create the fixed files in a user's directory. (GEN can also be used to expand the FCT table of an "old" CLPARS user.)

APPENDIX D

OLPARS "HELP" FUNCTION

The help file gives the user information, at his/her terminal, about an OLPARS subject or command. The "HELP" program is system dependent.

The "HELP" program prompts the user for input. The user may type in the name of an OLPARS subject or command for which (s)he desires information (help). Initial characters (substrings) of the command or subject name may also be typed. The user may type "ALL" for a list of available help files.

All the text, used to describe a specific OLPARS subject or command, is referred to as a "HELP" file. The character string typed by a user, in response to the "HELP" program prompt, will identify a "HELP" file to be accessed. All "HELP" files exist in the same directory and have the same RSX-11M filename extension (e.g. ".HLP").

In addition to accessing the "HELP" files, the "HELP" program accesses a file called "HELP.TXT" (the help dictionary), and a file called "HELPDIR.TXT." "HELP.TXT" contains an alphabetical listing of subjects and commands for which help is available. "HELPDIR.TXT" contains the device name and directory string needed to locate the "HELP" files. It also contains the RSX-11M filename extension for the "HELP" files. The "HELP" program uses the

information in "HELPDIR.TXT" to construct the complete RSX-11M filename for the "HELP" file to be accessed. The filenames "HELP.TXT" and "HELPDIR.TXT" are "built-in" to the "HELP" program. If for some reason one wanted to change these names, the "HELP" program would have to be modified. These two files reside in the OLPAR system directory, where the OLPARS tasks are located.

The existence of the "HELP.TXT" file makes it possible for the user to type a substring of a subject or command for which help is being requested. The "HELP" program searches the "HELP.TXT" file for the user-typed character string. If the character string is found, the complete subject or command name is gotten from "HELP.TXT" and is used to construct the "HELP" file name. Thus it is not necessary for the user to type the entire name. The user-typed character string should, however, be sufficiently long to be unique within "HELP.TXT." If the character string is not unique, help will be provided for the first subject or command in "HELP.TXT" whose initial characters match the user-typed character string. If the "HELP" program does not find the user-typed character string in "HELP.TXT," the character string itself will be used as the "HELP" filename and an attempt will be made to access the help file. Thus, there may be help available for subjects and commands which do not appear in "HELP.TXT." If the user wishes to access these help files, (s)he must type the complete "HELP" filename (not including the filename extension).

One special help file, the "ALL" help file, exists to provide the user with a list of subjects and commands for which help is available. The format of the "ALL" help file may vary. Subjects and commands may be listed together or separately, or alphabetically or categorically, depending on the system manager"s preference. One-line definitions for subjects and commands may be provided (e.g. since RSX-11M filenames are limited to nine characters, and it is difficult to describe a subject using only nine characters, "HELP" filenames pertaining to OLPARS subjects may not have meaning to a user. Therefore, a one-line description could accompany the subject name in the "ALL" help file).

The "HELP" function is designed to be flexible. The format of the help files (including the "ALL" help file) may vary and can be designated by the system manager. "HELP" files may be added to the help directory, at any time, through the use of a text editor. The system manager should obtain user feedback to determine which OLPARS subjects should have "HELP" files, and to decide upon a format for those files.

An example of filename construction using "HELP.TXT" and "HELPDIR.TXT."

HELP.TXT contains the following help filenames:

APPEND
CDEFAULT
DSCRMEAS
FILEIN
L2EIGV

HELPDIR.TXT contains the character string:
DBO:[313,46].HLP < ANY USEFUL COMMENTS MAY OCCUR HERE >

The user types the substring 'DSC' in response to the "HELP" command prompt. The "HELP" program searches the "HELP.TXT" file to find the string 'DSC'. "HELP" retrieves the "HELP.TXT" entry for 'DSCRMEAS'. Next, it reads the directory string from "HELPDIR.TXT" (a right bracket indicates the end of the directory string). It appends the name 'DSCRMEAS' to the directory string. Lastly, it reads the filename extension ".HLP" from "HELPDIR.TXT" and appends this to the directory-filename character string. The result is the complete filename:

DBO:[313,46]DSCRMEAS.HLP

This "HELP" file is printed at the user's terminal.

APPENDIX E

OLPARS Instrumentation Package

The instrumentation package is used as a debugging tool (instrument) for OLPARS programmers. The package consists of a series of subprograms that will enable a programmer to keep track of the entry and exit of a program, along with the values of specific variables within the program.

E.1 Philosophy

The programming staff at PAR has already had experience with the uses of an instrumentation package within a large conglomerate of programs. It was found that the instrumentation package was a very useful tool for debugging FORTRAN programs. Previous packages, however, did not make use of prior knowledge of a program's execution (i.e., the program's history). This meant that when the instrumentation was enabled, every program (in the larger system of programs) was traced, even if the program was fully tested and debugged. In certain cases, this also meant that a large amount of paper was used to print out the instrumentation tracing.

With this in mind, we thought it would be useful for a program, that has worked successfully for periods of time, to not participate in the instrumentation tracing while the package was enabled. To do this, each program's "history" would have to be monitored.

In the OLPARS instrumentation package, all programs are initially assumed to be "successful" programs. Since there is no "history" for new programs, this assumption must be made. However, if that program fails in its execution (i.e., it is not a successful program), its "history" will now indicate this fact. Further usages of the program will cause tracing to be seen (only while instrumentation is enabled) until the program has completed successfully some arbitrary number of times. Once the program completes this arbitrary number of successful executions, the tracing will again disappear.

With this method of program tracing, a programmer will not need to worry about programs that have good "histories". It will also keep the tracing to a minimum, thus, focusing a programmer's attention on the problem areas of a program. The programmer will not have to wade through piles of program tracing (a definite savings since paper is an expensive commodity).

E.2 Using the Instrumentation Package

To use the instrumentation package properly, a program should make reference to a few of the instrumentation programs in the following manner.

In the "main" program, before any disk files are opened,* a single call should be made to the instrumentation initialization routine (INSINI). This routine is used to enable or disable the instrumentation package for the current operation of the "main" program, initialize various instrumentation variables, and open two instrumentation information files. Before the "main" program exits, it should make a call to INSRET.

All subprograms** of the "main" program should make calls to the instrumentation subroutines 'INSPGM' and 'INSRET' after entry and upon exit to the subprogram, respectively. When the instrumentation packge is enabled, these two programs print out subprogram entrance and exit messages on the instrumentation listing, control the debug printout indentation, and keep track of the "history" of each subprogram.

----

* The reason for calling 'INSINI' before files are opened is explained later in "Some Notes about Instrumentation Package use on RSX-11M.

** There are a few programs that cannot make calls to the instrumentation package (because they are used by the instrumentation package; hence, infinite recursion would occur).

The above-mentioned steps are the only mandatory procedures to be adhered to for proper usage of the instrumentation package. The variable-printout (dump) calls to the instrumentation package may be done at any time after the initialization routine (INSINI) is called.

E.3 The "History" aspect of the CLPARS Instrumentation Package

Throughout the source code of the instrumentation package there is mentioned the concept of a program's "history." The stored information necessary to keep tally of a program's successful completion is called a program's "history."

Each CLPARS program (not found in the instrumentation package) will have its own history record which will be found in an CLPARS "history" file.* The history record will contain the name of the program, the length of that name, a tally requesting the number of times this program is to report its usage after failure, and an overflow link record pointer (see Figure E-1).

The "history" file is initially created by 'GEN', the CLPARS "fixed" files generating program. The header portion of the "history" file contains the record number of the first available overflow record in the file (see Figure E-2).

----

* The history file is one of the OLPARS user's "fixed" files. Under the RSX-11M operating system, the file is named 'HS.CLP'. Within OLPARS programs, the history file is referred to through its two letter mneumonic, 'HS'.

```
                |- .5 rel-|
 -------------------------------------------------------------
|          |            |            |                        |
| Overflow |            |    Name    |                        |
|   Link   |   Tally    |            |          Name          |
| Pointer  |            |   Length   |                        |
|          |            |            |                        |
 -------------------------------------------------------------
|-- 1 rel -|------ 1 rel * -----|----- 3 rels -----|
```

* Real element; under RSX-11M a 'rel' is composed of two-16 bit words

Figure E-1 OLPARS History file record

```
 -----------------------------------
| First Available overflow record  |
| number  (points to end of file)  |  Header
|==================================|
|          HISTORY RECORD          |
|----------------------------------|
|          HISTORY RECORD          |
|----------------------------------|
|                                  |
|                .                 |
|                .                 |  Primary
|                .                 |  Region
|                                  |
|----------------------------------|
|          HISTORY RECORD          |
|==================================|
|     OVERFLOW HISTORY RECORD      |
|----------------------------------|  Overflow
|                                  |  Region
|----------------------------------|
|                                  |
 -----------------------------------
```

Figure E-2 OLPARS History file

Initially, the "history" records have their link pointer, tally and name length set to zero.

The "history" file can be considered a "hash"* table. A program's "history" record is found by "hashing" the name of the program, using the resultant number as a pointer into the history file.

If two program names "hash" to the same history file record number (called colliding), then the link pointer of that record number is set to point to a record in the overflow area. This overflow record becomes the history record of the second program. Another collision to the same record number by another program would result in the link pointer of the first colliding program to point to a new overflow record. This process would be repeated for each new "hash" collision (see Figure E-3).

The "hashing" function being used is a slight modification of the division remainder algorithm found in most simple hashing routines. To keep collisions to a minimum (thus, file access), this algorithm requires that the number of entries in the table be prime. (Further details of the algorithm can be found in the OLPARS Software Reference Manual under the program name "HASH".)

----

* "Hashing" refers to the concept of mathematically combining the internal representation of a character string to create a single number. The resulting number is usually used as a pointer into some table.

```
                 ----------------
Header          | 23             |
                |================|
            1   | 20   |    A    |
                |------|---------|
            2   | 0    |         |
                |------|---------|
            3   | 0    |    F    |
                |------|---------|
            4   | 21   |    C    |
Primary         |------|---------|
Region      5   | 0    |         |
                |------|---------|
                      .
                      .
                      .
                |------|---------|
           19   | 0    |         |
                |================|
           20   | 0    |    B    |
                |------|---------|
Overflow   21   | 22   |    D    |
Region          |------|---------|
           22   | 0    |    E    |
                 ----------------
                (Link) (Program)
```

Programs in existence with history records are A,B,C,D,E, and F where:

Hash(A) = Hash(B)
Hash(C) = Hash(D)
Hash(C) = Hash(E)

That is, B collides with A, while D and E collide with C.

Figure E-3 Example of link pointer usage in OLPARS History file

The tally portion of the history record is used to decide whether or not any debug information is to be printed when the instrumentation package is enabled.

E.4 The Life of an CLPARS "History" Record

When the instrumentation package is enabled, the following steps occur for the subprogram "EXAMPL".

The subprogram "EXAMPL" is currently being executed. Its name is "hashed" and the resulting record number points to the history record for "EXAMPL". The record is read from the "history" file. It is found that the name length of the record is zero. Therefore, this is the first time "EXAMPL" has ever been used within the context of the current "history" file's existence. The name, "EXAMPL", along with its corresponding length, is placed within the record. the tally portion of the record is set to the current value of the tally (zero, because it is a new record) plus a specified tally increment. This information is saved within the instrumentation package and is also written into the history file. Consequently, if some error causes "EXAMPL" to abort its execution (or if it is forced to abort by the operating system), the tally within the "EXAMPL" history record is a nonzero positive number. The nonzero tally indicates that "EXAMPL" failed. The value of the tally represents the number of times (only while instrumentation is enabled) "EXAMPL" must be entered and exited successfully, before debug printout (trace) for "EXAMPL" will stop.

If "EXAMPL" has no "abortive bugs," just prior to exiting, the tally of its history record (found in the 'HISTRY' common area of the instrumentation package) will be effectively decremented by one (or set to zero when decrementing results in a negative tally). Then the record will be placed into the appropriate place within the "history" file, writing over the old version of the record.

E.5 Instrumentation Package Programs

The following two sections describe the program modules of the CLPARS instrumentation package available for programmer use. The first section describes the "necessary" instrumentation programs. An example "usage" is given for each program.

E.5.1 Control tracing and "History" maintenance programs

INSINI - Instrumentation package initialization, used to determine whether or not the instrumentation package is to be enabled or disabled for the current execution of the program calling INSINI.

```
CALL INSINI (Name)
```

where "Name" is a Hollerith string containing the name of the program calling INSINI. (Name should not exceed 9 characters.

Example: CALL INSINI('COMMCD')

INSPGM - Instrumentation package program entrance monitor, used to indicate that the currently executing program has just been called (i.e., a call to this program should be the FIRST executable statement in a subprogram).

CALL INSPGM (Name, PgmTyp)

where "Name" is a Hollerith string containing the name of the program calling INSPGM (name should not exceed 9 characters). "PgmTyp" is an integer specifying whether a program is of type "MAIN" (=0) or of type "SUBPROGRAM" (=1).

(BIG NOTE: 'INSINI' calls 'INSPGM', for the user with "PgmTyp" set to "MAIN". Therefore, the user should not call 'INSPGM' in the "main" program where 'INSINI' is used.)

Example: CALL INSPGM ('GNXTND',1)
CALL INSPGM ('EQUALA',SUBPGM)

INSRET - Instrumentation package exit monitor, used to indicate that the currently executing program is about to exit (i.e., a call to this program should be the LAST executed statement in the calling program).

Example: CALL INSRET

INSSET - Sets instrumentation package to a desired state, either "on" or "off". The previous state of the instrumentation package is returned to the calling program.

CLSTAT = INSSET (NwStat,Thresh)

where "NwStat" is the new state of the instrumentation package (true = ON, false = OFF); "Thresh" is an integer value representing the new debug printout threshold. Upon return, INSSET will return the old state of the instrumentation package as its function value and set "Thresh" to the previous threshold value of the package.

```
Example:  THRESH = 3
          CLSTAT = INSSET(.true., THRESH)
```

E.5.2 Variable trace programs

INSCHR - Prints out the value of a character string variable (i.e., prints out characters stored in an integer array, one character per integer).

```
CALL INSCHR (Name, Value, Length)
```

where "Name" is a Hollerith string containing the name of the variable to be printed (or some pertinent description); "Value" is the address of the variable to be printed; "Length" is the number of characters to be printed, when zero, the last character in the string must be followed by a zero, the end-of-string symbol.

```
Example:  CALL INSCHR ('CHAR', CHAR, 10)
          CALL INSCHR ('Tree Name', TNAM, 0)
```

INSDBL - Prints out the value of a double precision floating point variable.

CALL INSDBL (Name, Value)

where "Name" is a Hollerith string containing the name of the variable to be printed; "Value" is the address of the double precision variable to be printed.

Example: CALL INSDBL ('CORREL', CORREL)

INSFLT - Prints out the value of a single precision floating point variable.

CALL INSFLT (Name, Value)

where "Name" is a Hollerith string containing the name of the variable to be printed; "Value" is the address of the single precision variable to be printed.

Example: CALL INSFLT ('FAST', FAST)

INSINT - Prints out the value of the integer variable.

CALL INSINT (Name, Value)

where "Name" is a Hollerith string containing the name of the variable to be printed; "Value" is the address of the integer variable to be printed.

Example: CALL INSINT ('LENGTH OF NAME', I)

INSLOG - Prints out the value of a logical variable.

CALL INSLOG (Name, Value)

where "Name" is a Hollerith string containing the name of the variable to be printed; "Value" is the address of the logical variable to be printed.

Example: CALL INSLOG ('FIRST TIME FLAG', FRSTIM)

E.6 Some Notes About Instrumentation Package Use on RSX-11M

The instrumentation package has its own dedicated logical unit (for output) so that a programmer has the ability to direct the debug output to various computer peripheral devices (disk, line printer, terminal, etc.). Normally, under RSX-11M, the instrumentation package puts its output onto the "system" disk in the file 'INSTRU.DAT'. This can be modified during the task build (linking) stage (see ASC parameter in RSX-11M task build manual) or immediately prior to program execution (see REA command directive in the RSX-11M operator's procedures manual). Note also that if the device 'XAC:' is defined, 'INSTRU.DAT' will be placed on that device instead of the "system" (SYC:) device.

The logical unit used in CLPARS will be logical unit number two. This logical unit assignment will not be guaranteed, however, unless programmers call the instrumentation initialization routine (INSINI) before any other files are opened. In general, it would be a good practice to make a call to 'INSINI' the first executable statement in an CLPARS "main" program. This convention is necessary because logical unit assignment to files opened within a program are done on a first-come, first-served basis (Logical unit two happens to be the first unit in the list of available logical units). If the convention is not followed, a programmer who desired to redirect the debug output would have to figure out, for each program written, to which logical unit the instrumentation package was attached.

APPENDIX F

CLPARS RSX11M I/O Notes

F.1 Access to CLPARS User's Files

All access to any of the CLPARS User's Files will be obtained through the level II/level I manipulation and access routines (see Figure F-1).

F.1.1 (Fixed Files)

The level II manipulation routines, OPENFX and CREAFX, "know" the size of the headers and logical entries of all the fixed files. To open the files, these two routines call the level I routines, OOPEN and CCREAT, respectively. OOPEN and CCREAT retrieve/insert the system pointer (i.e., the system file name) to the fixed file from/to the File Code Table file, via the fixed file's file code. (The file codes for the fixed files are predefined to be the values 1 through 10). OOPEN and CCREAT then open the file and fill in the appropriate information in the File Access and Control Table (FACT). Once the FACT is filled in by the manipulation routines, the CLPARS fixed files can be accessed by the level II access routines, through the level I access routines, FGET and FPUT.

F.1.2 (Data and Logic Files)

The level II manipulation routines OPENTR and CREATR "know" the size of the headers of a user's data and logic files. They must, however, calculate the size of a logical entry in these files. The dimensionality of the data set, used to create the data or logic tree, is all that is necessary for this calculation. This information is stored in the Tree List and Logic List files, along with a tree's file codes, which are used to retrieve (from the File Code Table) the system names of the files that are used to simulate the OLPARS tree. The files are opened by either COPEN or CCREAT. From this point, operations continue as mentioned under "Fixed Files."

F.2 OLPARS Block I/O (File Access and Control Table) Notes

Currently, the file access and control table has the capability to have fifteen block I/O files open at one time. However, the entire FACT is not allocated at compile time. The reason for this was to save task space. The FACT was broken into two FORTRAN common areas. Everything but the Physical Record Buffer (PRBUFF) area was put in the common area labeled FACT. The Physical Record Buffer was placed in its own common area, called BLKBUF. The portion of the file access and control table that is in the FACT common area has enough space for the fifteen open files, as mentioned above. The BLKBUF common only has enough space allocated for one block (256 word) buffer. (These allocations are compile time allocations).

The BLKBUF area can be extended at task build (link) time by using the task builder EXTSCT directive. This gives a programmer the ability to specify, exactly, the number of block I/O buffers (maximum = 15) to create, according to the number of files (to be accessed using block I/O) that are opened at any one time. (See CLPARS command task building notes for RSX-11M, in section F.5).

The PRBUFF is separated from the FACT solely for the purpose of adjusting its size at task build time (because this was the only way it could be done).

F.3 CLPARS Record I/O Notes

Terminal and sequential file access is done through RSX-11's record I/O facility. RSX record I/O has a buffer region that corresponds to the CLPARS block I/O buffer, BLKBUF. The buffer is called $$FSR1. This region's size is also controlled at task build time, like the BLKBUF region. In this case, however, the region's size is controlled by the ACTFIL task build parameter, instead of the EXTSCT parameter.

F.4 CLPARS File I/O Communication Paths

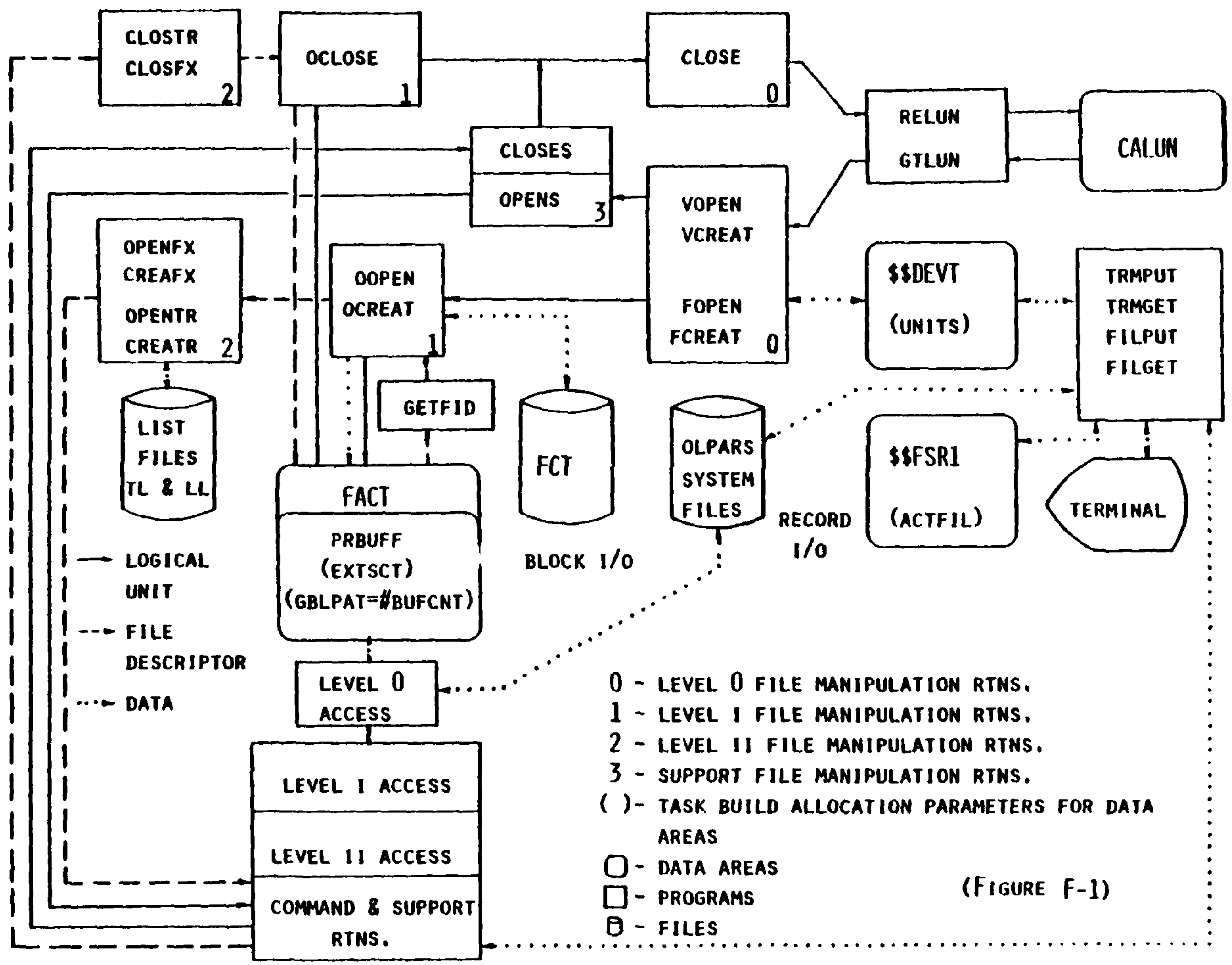
Figure F-1 shows the relationship of communication between CLPARS programs, files, and data areas. Following is an explanation of the figure.

F.4.1 (Logical Unit Allocation)

There is a global common area within CLPARS which contains controlling information about the allocation of logical unit numbers for both record and block I/O. The area is described as the currently available logical unit numbers table (CALUN). The table contains information about 15 logical unit numbers. Logical unit numbers are dispensed, upon request from the "get logical unit number program" (GTLUN), on a first come, first serve basis. GTLUN searches the table from beginning to end for a free unit. When one is found, GTLUN marks the unit as being used and returns the unit number to its calling program.

Logical units become "free" or available when the "release logical unit number program (RELUN) marks a unit for reuse.

The purpose behind this type of logical unit allocation was to free the programmer from "worrying" about which logical units could be used when opening a file.



OLPARS File I/O Paths on RSX11M

F.4.2 (Block I/O)

A command program that accesses an CLFARS user's fixed data or logic files must first call the appropriate level II "open" routine (i.e., OPENTR or CREATR for tree files, OPENFX or CREAFX for fixed files). If a tree file is being opened, information about the file must be retrieved from (or placed into when creating) the Tree (TL) or Logic List (LL) file. From this point, all remaining actions occur for both tree and fixed files. The level II "open" routines call the level I "open" routines. They, in turn, open the File Code Table and retrieve the "system" file name(s) of the file(s) to be opened. The level I open routines call the level zero* block I/O open routines (FOPEN, FCREAT) to actually open the "system" files. These routines request a logical unit number from the available pool of numbers (as mentioned above) and open the system file, which causes system file information to be placed into the device table ($$DEVT).**

----

* Level 0 programs mentioned here are RSX-11M assembly programs that communicate with the operating system directly for opening, closing, reading, writing, etc. of a file.

** The logical unit number is used as an index pointer into the device table.

The logical unit number is returned back to the level I "open" routines. The "open" routines locate a free file descriptor within the File Access and Control Table (FACT), through the "get file descriptor program" (GETFID). Once the file descriptor is obtained, the logical unit number, along with other appropriate information, is placed in the FACT. The level I "open" routines return to the level II "open" routines the value of the file descriptor, which is subsequently given to the upper level and support/command programs.

Once the files are opened, they can be accessed by the command programs through the level II, level I, level 0 path of programs. The level I access routines (FGET, FPUT) use the previously stored file access information in the FACT to direct the level zero routines. The level zero routines perform the actual block I/O access of the RSX-11M system files. The routines place/retrieve those blocks into/from the Physical Record Buffer area (PRBUFF) in the FACT.*

To close the CLPARS block I/O accessed file, one of the appropriate level II "close" routines (CLOSTR, CLOSFX) is called. The level II "close" routines pass the file descriptor to the level I close routine (OCLOSE). It obtains the logical unit of the file to be closed from the FACT and calls the level 0 close routine. The "system" file is closed (this clobbers information in $$DEVT)

----

* The PRBUFF is not actually contained "in" the FACT, but is considered part of it.

and the logical unit is marked "as available" in the logical unit pool.

F.4.3 (Record I/O)

A command program that accesses the user's terminal simply uses the terminal I/O package (TRMGET/TRMPUT). Internally, however, the terminal access programs can "sense" when their special logical unit** has been opened or not. When the unit is closed, TRMGET or TRMPUT makes a call to the terminal open routine (TRMOPN, found in the level 0 file manipulation routines). The "system" then places information about the file (terminal) into the device table ($$DEVT). The terminal routines now have the ability to access the terminal.

To access an RSX-11M sequential (variable length record) file, the command or support program must first open the file with the support file "open" routine (OPENS). The support routine calls a record I/O level zero "open" routine (VOPEN, VCREAT) to perform the actual "system" open.

As in block I/O "opens," the level zero "opens" requests a logical unit number to use and performs the "system open," which fills in the device table. This unit number is transferred directly back to the program that requested the file to be "opened". The sequential file access programs (FILGET, FILPUT) can

----

** Logical unit 1 has been reserved for terminal I/O and is not found in the currently available logical unit table.

now access the opened file. (Information is buffered in the $$FSR1 region for both the terminal access and sequential file access programs).

To close a sequential file (thus, flushing its buffer), the support file manipulation "close" routine must be called. This routine calls the level zero close routine, which closes the file and releases the logical unit number in use.

F.5 CLPARS Command Task Building Notes for RSX-11M

Disregarding the possibility of programs being "overlaid," the following RSX parameters will be used when task building (linking) an CLPARS command program. They are:

ACTFIL, EXTSCT, UNITS, GBLPAT, and ASG

where,

- ACTFIL - specifies the number of record I/O files to be open at any one time. This parameter regulates the size of the record I/O buffer region ($$FSR1).
- EXTSCT - is used to control the size of the block I/O buffer region (BLKBUF).
- UNITS - specifies the total number of files open at any one time (that's both record and block I/O files). This parameter controls the size of the logical unit device table region ($$DEVT).
- GBLPAT - is used to specify the number of block

buffers allocated. The use of this parameter will help protect the programmer when developing and debugging a new program. It will prevent him from clobbering (writing over) data areas outside the block buffer region (BLKBUF).

ASG - is used to assign logical units to particular devices at task build time.

The following is an example of a task build command file for building a non-overlaid CLPARS command. For more detail, refer to the RSX-11M task builder manual.

```
PROGRAM/CP/FP=PROGRAM,[313,32]CLPARSLIB/LB
;
;PROGRAM - name of the CLPARS command being built
;
;
;/CP - makes task checkpointable, so it can be swapped out of
;      memory in a multi-user environment.
;
;
;/FP - tells task builder that the floating point processor
;      is used by the task
;
;
;[312,32]OLPARSLIB/LB - the CLPARS subprogram library,
;              which happens to be in directory [313,32]
;
;
;
/
;There is one record I/O file open in PROGRAM.
 ACTFIL = 1
;
;There is one block I/O file open in PROGRAM.
 EXTSCT = BLKBUF:1000
 GBLPAT = PROGRAM:BUFCNT:1
;
;Note:  Both numbers in the above parameters are octal.  BLKBUF's
;       size is in bytes.  It should be extended (for more block
;       I/O files to be opened) in increments of 1000 (8) or
;       512 (10).  If neither parameter is used above, it should
;       be known that there is one block buffer allocated at
;       compile time (i.e., BLKBUF has 512 (10) bytes allocated
;       and BUFCNT is set to 1).
;
;
;
; There are a total of two files opened at any one time in PROGRAM.
; Currently the maximum number allowed is 15.
 UNITS = 2
;
; Logical unit 1 should ALWAYS be assigned to the terminal device.
; The remaining logical units are assigned to the "system" device.
 ASG = TI:1, SY:2
//
```

APPENDIX G

OLPARS Programmer Aides

The following pages contain information on how to use the OLPARS programmer aides. The current programmer's aides are GEN, INSHSP, MAKOPT, CDTDMP, and OLTDMP. GEN is used to generate a new OLPARS user directory (see APPENDIX C), expand an old user-directory, or give a list of the number of trees a user is allowed. INSHSP can be used to activate or deactivate debug print-out of specific programs when program instrumentation is present in the task and turned on. MAKOPT is used to create the OLPARS option file (see Appendix B). CDTDMP and OLTDMP can be used to dump the structural content of an OLPARS data or logic tree, respectively.

The format of the aid descriptions is identical with the OLPARS User's Manual format.

COMMAND NAME: GEN

CATEGORY: Programmer's Aide (system dependent)

FUNCTIONAL DESCRIPTION:

The GEN utility creates all the necessary CLPARS 'fixed' files that are required by an CLPARS user. File headers are appropriately initialized. GEN also sets up the File Code Table, which specifies the total number of trees (data/logic) that an CLPARS user is allowed to have in his directory. The user has the additional options of expanding his tree capacity (allow for more trees) and of printing out his tree capacity (list out the number of trees allowed, the number of trees currently existing, and the remaining number of trees allowed).

USER INTERACTION:

There are three possible ways to run gen:

1) GEN<cr>

2) GEN 'switch' <cr>

3) RUN GEN<cr>

1 and 2 above assume that the GEN task has been installed. (INS GEN/TASK=...GEN)

GEN takes the following switches:

/NW:n create a new user with a tree capacity of n

/EX:n expand an old user's tree capacity by n

/LI print out a user's tree capacity

EXAMPLE(S):

Three examples of GEN sessions will be given, one for each method of running GEN.

1) Session when user types GEN<cr>

```
/GEN<cr>/

GEN>/ /LI<cr>/

USER'S TREE CAPACITY = 10

THE NUMBER OF TREES EXISTING IN THE USER'S DIRECTORY = 0

THE REMAINING NUMBER OF TREES THAT THE USER IS ALLOWED
= 10

GEN>/<cr>/
```

2) Session when user types GEN 'switch' <cr>

```
/GEN /EX:20<cr>/  (the outer-most slashes mark user
                    response)
>
```

3) Session when user types RUN GEN<cr>

```
/RUN GEN<cr>/

TO EXPAND A USER'S TREE CAPACITY, TYPE /EX:N
WHERE N IS THE NUMBER OF TREES BY WHICH TO
EXPAND A USER'S TREE CAPACITY

TO CREATE A NEW USER, TYPE /NW:N
WHERE N IS THE NEW USER'S TREE CAPACITY

TO PRINT OUT A USER'S TREE CAPACITY, TYPE /LI

//NW:50<cr>/  (the outer-most slashes mark user response)

>
```

(NOTES: **

If the user types the switches to create or expand an CLPARS directory (/EX:n or /NW:n), and GEN completes successfully, no messages are printed at the terminal.

Running GEN by typing GEN<cr> allows the user to run as many GEN options as he wishes. After each option has been executed, the prompt 'GEN>' is put out at the user's terminal. He can then type in another option (switch), or type a carriage return or a control-Z to exit. Running GEN using either of the other two methods (RUN GEN<cr>, or GEN 'switch' <cr>) only allows for one option to be executed. After execution of the option, GEN halts and the MCR prompt (>) is put up at the user's terminal.

In the examples above, the expression / /LI/ means that the user typed " /LI".
Typing a space before a switch or before a carriage return is optional.

***)

COMMAND NAME: INSHSP

CATEGORY: Programmer's Aide

FUNCTIONAL DESCRIPTION:

INSHSP (Instrumentation History file Patch) can be used by an CLPARS programmer to activate or deactivate instrumentation for a selected set of CLPARS subprograms. This can help keep the amount of printed information in the instrumentation output to a minimum.

USER INTERACTION:

User is asked:

1. for the name of the file containing names of programs to activate or deactivate in instrumentation.

2. to 'activate' or 'deactivate' given programs

EXAMPLE(S):

A file (OLPPGMS.TXT) has been created containing the names TLSRCH, GARBND, EQUALA, and OPENTR on separate lines (upper case letters are important). The program CDEFAULT has been executed to set the instrumentation 'ON' and set the instrumentation threshold to five (this will silence the other succesfully executed programs).

```
TYPE IN FILENAME CONTAINING NAMES OF
PROGRAMS TO ACTIVATE/DEACTIVATE - /OLPPGMS.TXT/
ACTIVATE OR DEACTIVATE (A/D)? /A/
```

(PROMPT NOTES: ***************************************
If the next CLPARS command executed enters any of the programs 'activated', then instrumentation will appear for those programs. Note: instrumentation may also appear for any programs that exceed the current instrumentation threshold requirements other than those specified thru 'activation'.

The program activation will last at least five times the number of times the program name appears in the activation file. So, if a program's name appears once in the activation file, and it is successfully executed five times, then instrumentation will cease for that program.
***)

COMMAND NAME: MAKOPT

CATEGORY: Programmer's Aide (system dependent)

FUNCTIONAL DESCRIPTION:

MAKOPT creates the OLPARS option file (OPTION.CLP) within the OLPARS directory. MAKOPT uses the option text file (OPTION.TXT) as an input file. This program should be used only by OLPARS programmers/maintainers to create the proper option file. See Appendix E of this manual for a description of OPTION.CLP and OPTION.TXT and for information on future Option File Maintenance.

USER INTERACTION: NONE

(NOTES: **

If for some reason MAKOPT cannot create the option file, an appropriate error message will be printed at the user's terminal, and the program will exit. If the program completes successfully, the number of program names processed is printed at the user's terminal.

**)

COMMAND NAME: ODTDMP

CATEGORY: Progra[illegible]er's Aide

FUNCTIONAL DESCRIPTION:

The ODTDMP (OLPARS Data Tree Dump) utility is used to check the structural integrity of an OLPARS data tree.

USER INTERACTION:

User types 'RUN <pathname>ODTDMP' to initiate the program. <pathname> is whatever is needed to locate the program.

Once the program is running, the user is asked for the name of the OLPARS data tree to be dumped.

NOTE: To use this program properly, you must be 'in' the OLPARS user's directory where the data tree to be dumped is located.

EXAMPLE(S):

The tree 'nasa1' is going to be dumped.

1) RUN DB:[313,40]ODTDMP

2) the program prompts the user with:

OLPARS DATA TREE NAME - /nasa1/

3) results in:

STRUCTURAL DESCRIPTION OF OLPARS TREE nasa1

```
 8      NODES IN TREE
12      IS VECTOR DIMENSIONALITY
 0      NODES IN FREE LIST
 1      IS SENIOR NODE SLOT IN ENTRY TABLE
```

ENTRY TABLE

	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

		COUNTS					POINTERS					
NODE	NAME	VEC	VCP	LNC	KID	NL	P	S	FC	FLN	LNL	LNB
1	****	422	0	7	7	1	-	-	s	s	-	-
2	soy	61	1	0	0	2	*	c	-	-	c	-
3	corn	60	62	0	0	2	*	o	-	-	o	s
4	oats	65	122	0	0	2	*	w	-	-	w	c
5	weat	60	187	0	0	2	*	C	-	-	C	o
6	Clov	62	247	0	0	2	*	a	-	-	a	w
7	alfa	54	309	0	0	2	*	r	-	-	r	C
8	rye	60	363	0	0	2	*	-	-	-	-	a

STRUCTURAL PICTURE FOR nasa1

```
****
    soy
    corn
    oats
    weat
    Clov
    alfa
    rye
```

(PROMPT NOTES: ***************************************

If you wish to exit from CDTDMP at the tree name prompt, enter an empty line.

Following is an expansion of the abbreviated headings in the counts and pointers table seen in the previous example.

VEC - the number of vectors that 'lie' at the given node.

VCP - the position of the vectors in the Tree Vector file for the given lowest node.

LNC - lowest node count, the number of lowest nodes beneath the given node.

KID - the number of children (kids) for the given node.

NL - node level of the given node.

P - parent of the given node.

S - sibling of the given node.

FC - first child of the given node.

FLN - first lowest node of the given node

LNL - lowest node link, node following the given node in the lowest node chain.

LNB - lowest node back link, node preceding the given node in the lowest node chain.

**)

COMMAND NAME: OLTDMP

CATEGORY: Programmer's Aide

FUNCTIONAL DESCRIPTION:

The OLTDMP (OLPARS Logic Tree Dump) utility is used to check the structural integrity of an OLPARS logic tree.

USER INTERACTION:

User types 'RUN <pathname>OLTDMP' to initiate the program. <pathname> is whatever is needed to locate the program.

Once the program is running, the user is asked for the name of the OLPARS logic tree to be dumped.

NOTE: To use this program properly, you must be 'in' the OLPARS user's directory where the logic tree to be dumped is located.

EXAMPLE(S):

The tree 'EXAMPLE' is going to be dumped.

1) RUN DB:[313,40]OLTDMP

2) the program prompts the user with:

OLPARS LOGIC TREE NAME - /EXAMPLE/

3) results in:

STRUCTURAL DESCRIPTION OF CLPARS LOGIC TREE EXAMPLE

```
DESIGN DATA SET  - EXAMPLE
  DIMENSIONALITY - 4
  CLASS COUNT    - 4
NODES
  NEXT AVAILABLE(8)  AT END OF FILE(9)  IN USE(6)
  (NODES IN FREE LIST 8 7 )
CURRENT LOGIC NODE - 4
REASSOCIATED NAMES - NO
LV ENTRY SIZE      - 12
INCOMPLETE NODE CT.- 4
```

DDS CLASSES PRESENT IN TREE, WITH A-PRIORI PROBABILITY

ABC 0.254 DEFG 0.254 CAT 0.288 SAND 0.203

NODE	LOGIC	CREATING COMAND	NL	KID	PP	FC	SB	LVP	RJP	OCNM	RCNM	MLF	NCF	PR.
1	GROUP	L1EIGV	0	3	0	2	0	1	0	----	----	0	4	ADCS
2	GROUP	L1EIGV	1	2	1	5	3	3	0	----	----	0	3	ADC
3	INCPLT	L1EIGV	1	0	1	0	4	0	0	----	----	0	2	AD
4	INCPLT	L1EIGV	1	2	1	7	0	0	0	----	----	0	3	ADS
5	INCPLT		2	0	2	0	6	0	0	----	----	0	3	ADC
6	INCPLT		2	0	2	0	0	0	0	----	----	0	2	AD

STRUCTURAL PICTURE FOR EXAMPLE

```
1 (SENIOR NODE)
   2
      5
      6
   3
   4
```

(PROMPT NOTES: **

If you wish to exit from OLTDMP at the tree name prompt, enter an empty line.

Following is an expansion of the abbreviated headings in the counts and pointers table seen in the previous example.

NL - node level (with senior node starting at zero)

KID - number of children below a given node

PP - parent of the given node.

FC - first child of the given node.

SB - sibling of the given node.

LVP - decision logic vector pointer to LV file

RJP - reject logic pointer to LV file

OCNM - original data class name (should have value when only one class exists at the given logic node)

RCNM - reassociated class name (should have value when only one class exists at the given logic node)

MLF - modified logic flag

NCP - number of classes present

PR - class symbols of classes present at given node

***)

APPENDIX H

CLPARS Parameter Limits

This appendix defines parameter limits (minimum and maximum values) for parameters used in CLPARS. These parameters can be utilized by including the appropriate declaration file in the source program. The parameters are first grouped according to alphabetical order. Their 'location' (the file in which the parameter can be found) is determined by attaching a file <type> name of '.DCL' to the name given in the list (E.G., SCREEN.DCL). The second grouping lists the parameters according to the file in which they reside.

Parameter	Value	Location	Description
A	7	SCREEN	index to the x-coordinate of the lower left point of the display rectangle
ACCESS	0	DISPLAY	indicator that an item is to be read from a display file header
ALIFLG	7	LIHDR	alias flag
ALPHA	1	CHARTYPE	used when characters in name can only be alphabetic
ALPHNM	2	CHARTYPE	used when characters in name can be alpha-numeric
ANGLE	0	DISPLAY	dummy angle parameter for btmap routines

Parameter	Value	Location	Description
ANYCHR	3	CHARTYPE	used when any printable ASCII character (PASCII), DIGIT, or LETTER is allowed in an identifier
APPEND	2	STANDARD	open-for-writing-at-the-end-of-the-file indicator
B	8	SCREEN	index to the y-coordinate of the lower left point of the display rectangle
BMPSIZ	2	BITMAP	the size of the classes present bitmap on PDP-11 machines (when the maximum number of classes is equal to 50)
BPET	12	TIFILE	the back pointer to the entry table slot number of a node
BUFSIZ	128	FACT	physical record buffer size (in real elements)
C	9	SCREEN	index to the x-coordinate of the upper right point of the display rectangle
CAPSIZ	12	TIFILE	the size of the counts and pointers region of a TI node
CLPBMS	5	RNKORD	class pair by measurement type of ranking
CLSBMS	4	RNKORD	class by measurement type of ranking
CLSDBN	4	LOGTYPE	closed decision boundary logic nodes
CLUSTR	1	DISPLAY	the display code for a two-space cluster plot
CM	1	FCDS	communications file code (FCD)
CMDLEN	10	STANDARD	maximum character length of a command
CMHNE	97	LENGTH	the length (in real elements) of the communications file header

OLPARS Parameter Limits -- H
Alphabetized

Parameter	Value	Location	Description
CMLENE	0	LENGTH	the length (in real elements) of a logical entry of the communications file
CONMAT	4	DISPLAY	the display code for a confusion matrix display
D	10	SCREEN	index to the y-coordinate of the upper right point of the display rectangle
DATA	1	STANDARD	data tree indicator
DC	1	RNKORD	the index in the DI file array for the display code
DDSDIM	1	LIHDR	design data set dimensionality
DDSNLN	2	LIHDR	number of lowest nodes in design data set
DELETE	1	STANDARD	creating-a-temporary-file indicator
DI	4	FCDS	display information file code (FCD)
DIGIT	2	CHARTYPE	means character is a digit
DIHNE	1	LENGTH	the length (in real elements) of the display information file header
DILENE	1	LENGTH	the length (in real elements) of a logical entry of the logic list file
DINITM	17	DISPLAY	the number of DI file header elements referenced by subroutines DIGHDI and DIPHDI
DS	19	SCREEN	index to the distance between succesive one space 'macro' display base lines
DSPCHR	8	RNKORD	the index in the DI file array for the ordered list of display characters

Parameter	Value	Location	Description
DV	5	FCDS	display value file code (FCD)
DVHNE	1	LENGTH	the length (in real elements) of the display value file header
DVLENE	1	LENGTH	the length (in real elements) of a logical entry of the display value file
DVNITM	4	DISPLAY	the number of DV file header elements referenced by subroutines DVGHDR and DVPHDR
E	15	SCREEN	index to the minimum x-coordinate of a one space base line
ENMEAN	17	TIFILE	the first element number of the means in the TI file
EOF	-1	STANDARD	end-of-file code
EOS	0	STANDARD	end-of-string indicator for integers
ERROR	-1	STANDARD	the standard error value throughout CLPARS
EXMLIM	150	LIMITS	the maximum number of measurements allowed in a vector in excess measurement mode
F	16	SCREEN	index to the minimum y-coordinate of a one space base line
FCP	7	TIFILE	the code for accesssing the first child pointer of a node
FIXED	5	FTYPE	the fixed file type indicator
FKIDND	4	LISTRC	index to the first child pointer of the current logic node

CLPARS Parameter Limits -- H
Alphabetized

Parameter	Value	Location	Description
FLNP	8	TIFILE	the code for accessing the first lowest node pointer
FNAMSZ	13	FACT	size (in words) of system file name
FNMLEN	34	STANDARD	the maximum length of an RSX-11 filename
FREE	0	CALUN	value to indicate that a 'LUN' is free
FRELST	13	TIFILE	pointer to the next "inactive" member in the free list (one past the current member)
G	17	SCREEN	index to the maximum x-coordinate of a one space base line
GRPLOG	2	LOGTYPE	group logic nodes (1-space, 2-space, boolean)
HC	6	SCREEN	index to the number of display units in character height
HD	4	SCREEN	index to the number of display units in display height
HDRLEN	3	OPTION	the length of a header record in OPTION.CLP
HEADER	0	STANDARD	header indicator value to the 'FGET' and 'FPUT' routines
HELP	-2	STANDARD	the standard help value response from 'TRMGET'
HGT	0	DISPLAY	dummy height parameter for btmap routines
HS	10	FCDS	history file code (FCD)
HSHNE	1	LENGTH	the length (in real elements) of the history file header

Parameter	Value	Location	Description
HSLENE	5	LENGTH	the length (in real elements) of a logical entry of the history file
HSN	2	SCREEN	index to the number of display units in screen height
HSTBSZ	503	HISTRY	historic file table size (important: prime no.)
IGNORE	-1	DISPLAY	indicator that an item is not to be read/written from/to a display file header
INCPLT	0	LOGTYPE	incomplete logic nodes
INDI	0	DISPLAY	dummy indicator parameter for btmap routines
INITHR	5	HISTRY	initial threshold value for debugging information
LCH	13	SCREEN	index to the number of characters per line
LETTER	1	CHARTYPE	means character is a letter (either UPPER/lower case)
LEVEL1	0	INSTRU	initial debugging printout level
LIFILE	3	FTYPE	the logic information file type indicator
LIHENE	272	LIFILE	the LI header number of elements
LILENE	22	LIFILE	the LI logical entry number of elements
LL	3	FCDS	logic list file code (FCD)
LLHNE	2	LENGTH	the length (in real elements) of the logic list file header
LLLENE	25	LENGTH	the length (in real elements) of a logical entry of the logic list file

CLPARS Parameter Limits -- H
Alphabetized

Parameter	Value	Location	Description
LNEP	10	TIFILE	the code for accessing the lowest node back pointer of a node pointer of a node
LNCRNT	6	LIHDR	current logic node
LNLP	9	TIFILE	the code for accessing the lowest node link
LOGIC	2	STANDARD	logic tree indicator
LOWEST	6	LOGTYPE	all lowest logic nodes in logic tree (both complete and incomplete) (lowest means, 'no children')
LSTRSZ	7	LISTRC	logic tree structure size
LUNPTR	1	CALUN	'LUN' pointer portion of 'Luntbl'
LVFILE	4	FTYPE	the logic value file type indicator
LVHENE	3	LVFILE	the LV header number of elements
LVLMIN	12	LVFILE	the minimum length of a logical entry in the LV file
LVLOGK	6	LISTRC	Logic value file pointer index to decision logic for given logic node
LVNELM	8	LIHDR	number of elements in an LV file entry
LVRJCT	7	LISTRC	Logic value file pointer index to reject strategy logic for given logic node
MACRO	5	DISPLAY	the display code for a one-space macro plot
MAIN	0	INSGLBL	signifies to INSPGM that the calling program was a c 'main' program

Parameter	Value	Location	Description
MAXBDY	2	LIMITS	maximum number of data partitions (boundaries) allowed to be made on an CLPARS data set
MAXBIN	50	LIMITS	the maximum number of "bins" allowed in a one-space display
MAXBPT	6	LIMITS	maximum number of points allowed in a two space boundary
MAXBUF	1	FACT	the number of physical record buffers is originally set to one but will be extended at task build time
MAXCLS	50	LIMITS	the maximum number of data classes allowed within CLPARS
MAXDIM	50	LIMITS	the maximum vector dimensionality in CLPARS (of vectors not in excess measurement mode)
MAXFIL	15	FACT	maximum number of files that can be open simultaneously
MAXHSP	20	HISTRY	maximum size that the historic stack pointer can attain
MAXLIN	133	STANDARD	the maximum length of an I/O character string
MAXLUN	15	CALUN	maximum number of available logical unit numbers
MAXMTR	10	SMFILE	the maximum number of matrices allowed to be stored in the SM file
MAXNAM	9	HISTRY	maximum number of characters allowed in an CLPARS command (program) name
MAXOPT	15	OPTION	the maximum number of options a program can have

Parameter	Value	Location	Description
MAXPGM	106	OPTION	the maximum number of "program names" that can appear in the OPTION.TXT file
MAXSEG	10	DSCRMTHR	the total number of segments possible in two user specified boundaries
MAXSEG	10	GP2BLK	the total number of segments possible in two user specified boundaries
MAXVEC	32767	LIMITS	largest number of vectors allowed in an OLPARS data tree (system dependent on integer size, 2**(number of bits in an integer - 1) -1)
MBCLPR	3	RNKORD	measurement by class pair type of ranking
MBYCLS	2	RNKORD	measurement by class type of ranking
MENUMX	20	OPTION	the maximum number of menus per program name
MENVEC	-1	DISPLAY	the vector id in the DV file indicating that the vector is the mean vector
MESVEC	1	PVFILE	the entry number of the measurement vector in the [illegible] file
MICRO	6	DISPLAY	the display code for a one-space micro plot
MRW	11	SCREEN	index to the number of rows in the cluster plot grid
NC	18	SCREEN	index to the maximum number of classes that can be displayed on a one space 'macro' display at any one time

Parameter	Value	Location	Description
NCL	12	SCREEN	index to the number of columns in the cluster plot grid
NDIMEN	3	RNKORD	the index in the LI file array for the dimensionality
NDISP	6	LIMITS	the maximum number of different display types found in CLPARS
NEW	1	STANDARD	new-file-to-be-opened indicator
NFIXFL	10	STANDARD	the number of CLPARS fixed files
NINCLN	9	LIHDR	number of incomplete logic nodes
NL	4	TIFILE	the code for accessing the node level of a node
NLNODE	2	TIFILE	the code for accessing the number of lowest nodes beneath a node
NMVLOG	3	LOGTYPE	nearest mean vector logic nodes
NNUSED	5	LIHDR	number of nodes actively used in logic tree
NODENL	2	LISTRC	number-of-nodes-at-next-level index (number of children below the current node)
NODLEN	4	STANDARD	maximum length of a node name
NODLVL	1	LISTRC	index to the current logic node's level in the logic tree
NOKIDS	3	TIFILE	the code for accessing the number of children at a node
NOSCRN	19	SCREEN	the number of screen coordinates in the CM file

CLPARS Parameter Limits -- H
Alphabetized

Parameter	Value	Location	Description
NOTSET	0	DISPLAY	the display code indicating that the display is not set
NOVECS	1	TIFILE	the code for accessing the number of vectors at a node
NOZOOM	0	DISPLAY	the flag indicating that zooming is not in effect
NRSTNB	5	LOGTYPE	nearest neighbor logic nodes
NUMCLS	4	RNKORD	the index in the DI file array for the number of classes
NXTAVN	3	LIHDR	next available node entry in LI file
NXTEND	4	LIHDR	next open node entry at end of file
OFF	0	DISPLAY	the display flag value in a DI file logical entry that means the class should not be displayed OR the screen coordinate flag in the DV file header that means the screen coordinates have not been computed
OFNLEN	10	OPTION	the length of the option file name (OPTION.CLP)
OLD	0	STANDARD	old-file-to-be-opened indicator
ON	1	DISPLAY	the display flag value in a DI file logical entry that means the class should be displayed OR the screen coordinate flag in the DV file header that means the screen coordinates have already been computed
ONESPC	1	DISPLAY	one-space indicator
OPTLE	9	OPTION	the option file (OPTION.CLP) logical entry size =((MAXOPT+1)/2) +1

Parameter	Value	Location	Description
OPTNUM	2	RNKORD	the index in the DI file array for the option number of the creating command
OTHER	4	CHARTYPE	means character is not a LETTER, DIGIT, or PASCII character
OVRALL	1	RNKORD	overall type of ranking indicator
PAIRWS	1	LOGTYPE	pairwise logic nodes
PASCII	3	CHARTYPE	means character is a visible (printable) ASCII char. other than a DIGIT or a LETTER
PGMLEN	10	OPTION	the maximum number of characters allowed in a program name
PP	5	TIFILE	the code for accesssing the parent pointer of a node
PRNTND	3	LISTRC	index to the parent node of the current logic node
PV	6	FCDS	projection vector file code (FCD)
PVHNE	18	LENGTH	the length (in real elements) of the projection vector file header
PVLENE	1	LENGTH	the length (in real elements) of a logical entry of the projection vector file
PVNITM	6	DISPLAY	the number of PV file header elements referenced by subroutines PVGHDR and PVPHDR
QUIT	-1	STANDARD	the standard exit value response from 'TRMGET'
READ	0	STANDARD	read indicator to 'OPEN' routines

Parameter	Value	Location	Description
RECTAN	0	DISPLAY	the indicator that rectangular scaling is to be used for a two-space display
RNKOPT	5	RNKORD	the index in the DI file array for the rank order option
RNKORD	3	DISPLAY	the display code for a rank order display
RNKPRM	6	RNKORD	the index in the DI file array for the rank option parameters
S1	7	FCDS	scratch1 file code (FCD)
S1HNE	1	LENGTH	the length (in real elements) of the scratch1 file header
S1LENE	1	LENGTH	the length (in real elements) of a logical entry of the scratch1 file
S1NITM	4	DISPLAY	the number of S1 file header elements referenced by subroutines S1GHDR and S1PHDR
SAVE	0	STANDARD	creating-a-permanent-file indicator
SBTMAP	21	BITMAP	the element number in an LI file entry at which the classes present bitmap starts
SC2NUM	13	SCREEN	number of elements of screen coordinate infor- mation used for two-space displays
SCATTR	2	DISPLAY	the display code for a two-space scatter plot
SIBLND	5	LISTRC	index to the sibling node of the current logic node
SM	9	FCDS	saved matrix file code (FCD)

Parameter	Value	Location	Description
SMHNE	114	LENGTH	the length (in real elements) of the saved transformation matrix file header
SMLENE	51	LENGTH	the length (in real elements) of a logical entry of the saved transformation matrix file
SP	6	TIFILE	the code for accessing the sibling pointer of a node
SQUARE	1	DISPLAY	the indicator that square scaling is to be used for a two-space display
SRTFLG	7	RNKCRD	the index in the DI file array for the sort sort flag
SUBPGM	1	INSGLBL	signifies to INSPGM that the calling routine was c a subprogram
SV	8	FCDS	saved vector file code (FCD)
SVHNE	2	LENGTH	the length (in real elements) of the saved vector file
SVLENE	59	LENGTH	the length (in real elements) of a logical entry of the saved vector file
TABSIZ	100	TIFILE	the maximum size of a TI entry table
TIFILE	1	FTYPE	the tree information file type indicator
TIHSIZ	105	TIFILE	the size (in elements) of the TI file header
TL	2	FCDS	tree list file code (FCD)
TLHNE	2	LENGTH	the length (in real elements) of the tree list file header
TLLENE	12	LENGTH	the length (in real elements) of a logical entry of the tree list file

CLPARS Parameter Limits -- H
Alphabetized

Parameter	Value	Location	Description
TRELEN	8	STANDARD	maximum length of a treename
TVFILE	2	FTYPE	the tree vector file type indicator
TVHENE	11	TVFILE	the tree vector file's header entry size
TWOSPC	2	DISPLAY	two-space indicator
USAGE	2	CALUN	usage portion of 'Luntbl'
USED	1	CALUN	value to indicate that a 'LUN' is used
VP	11	TIFILE	the code for accessing the vector pointer to the vectors of a node in the TV file
VRNLEN	30	INSGLBL	length of a Hollerith string representing a variable name
WC	5	SCREEN	index to the number of display units in character width
WD	3	SCREEN	index to the number of display units in display width
WID	0	DISPLAY	dummy parameter for btmap routines
WRITE	1	STANDARD	write indicator to 'OPEN' routines
WS	1	SCREEN	index to the number of display units in screen width
ZCOM	1	DISPLAY	the flag indicating that zooming is in effect

OLPARS Parameter Limits -- H
According to File Location

Parameter	Value	Location	Parameter	Value	Location
BMFSIZ	2	BITMAP	MAXSEG	10	DSCRMTHR
SBTMAP	21	BITMAP			
			BUFSIZ	128	FACT
FREE	0	CALUN	FRAMSZ	13	FACT
LUNPTR	1	CALUN	MAXBUF	1	FACT
MAXLUN	15	CALUN	MAXFIL	15	FACT
USAGE	2	CALUN			
USED	1	CALUN	CM	1	FCDS
			DI	4	FCDS
ALPHA	1	CHARTYPE	DV	5	FCDS
ALPHNM	2	CHARTYPE	HS	10	FCDS
ANYCHR	3	CHARTYPE	LL	3	FCDS
DIGIT	2	CHARTYPE	PV	6	FCDS
LETTER	1	CHARTYPE	S1	7	FCDS
OTHER	4	CHARTYPE	SM	9	FCDS
PASCII	3	CHARTYPE	SV	8	FCDS
			TL	2	FCDS
ACCESS	0	DISPLAY			
ANGLE	0	DISPLAY	FIXED	5	FTYPE
CLUSTR	1	DISPLAY	LIFILE	3	FTYPE
CONMAT	4	DISPLAY	LVFILE	4	FTYPE
DINITM	17	DISPLAY	TIFILE	1	FTYPE
DVNITM	4	DISPLAY	TVFILE	2	FTYPE
HGT	0	DISPLAY			
IGNORE	-1	DISPLAY	MAXSEG	10	GP2BLK
INDI	0	DISPLAY			
MACRO	5	DISPLAY	HSTBSZ	503	HISTRY
MENVEC	-1	DISPLAY	INITHR	5	HISTRY
MICRO	6	DISPLAY	MAXHSP	20	HISTRY
NOTSET	0	DISPLAY	MAXNAM	9	HISTRY
NOZOOM	0	DISPLAY			
OFF	0	DISPLAY	MAIN	0	INSGLBL
ON	1	DISPLAY	SUBPGM	1	INSGLBL
ONESPC	1	DISPLAY	VRNLEN	30	INSGLBL
PVNITM	6	DISPLAY			
RECTAN	0	DISPLAY	LEVEL1	0	INSTRU
RNKORD	3	DISPLAY			
S1NITM	4	DISPLAY	CMHNE	97	LENGTH
SCATTR	2	DISPLAY	CMLENE	0	LENGTH
SQUARE	1	DISPLAY	DIHNE	1	LENGTH
TWOSPC	2	DISPLAY	DILENE	1	LENGTH
WID	0	DISPLAY	DVHNE	1	LENGTH
ZOOM	1	DISPLAY			

CLPARS Parameter Limits -- H
According to File Location

Parameter	Value	Location
DVLENE	1	LENGTH
HSHNE	1	LENGTH
HSLENE	5	LENGTH
LLHNE	2	LENGTH
LLLENE	25	LENGTH
PVHNE	18	LENGTH
PVLENE	1	LENGTH
S1HNE	1	LENGTH
S1LENE	1	LENGTH
SMHNE	114	LENGTH
SMLENE	51	LENGTH
SVHNE	2	LENGTH
SVLENE	59	LENGTH
TLHNE	2	LENGTH
TLLENE	12	LENGTH
LIHENE	272	LIFILE
LILENE	22	LIFILE
ALIFLG	7	LIHDR
DDSDIM	1	LIHDR
DDSNLN	2	LIHDR
LNCRNT	6	LIHDR
LVNELM	8	LIHDR
NINCLN	9	LIHDR
NNUSED	5	LIHDR
NXTAVN	3	LIHDR
NXTEND	4	LIHDR
EXMLIM	150	LIMITS
MAXBDY	2	LIMITS
MAXBIN	50	LIMITS
MAXBPT	6	LIMITS
MAXCLS	50	LIMITS
MAXDIM	50	LIMITS
MAXVEC	32767	LIMITS
NDISP	6	LIMITS
FKIDND	4	LISTRC
LSTRSZ	7	LISTRC
LVLOGK	6	LISTRC
LVRJCT	7	LISTRC
NODENL	2	LISTRC
NODLVL	1	LISTRC
PRNTND	3	LISTRC
SIBLND	5	LISTRC
CLSDBN	4	LOGTYPE
GRPLOG	2	LOGTYPE
INCPLT	0	LOGTYPE
LOWEST	6	LOGTYPE
NMVLOG	3	LOGTYPE
NRSTNB	5	LOGTYPE
PAIRWS	1	LOGTYPE
LVHENE	3	LVFILE
LVLMIN	12	LVFILE
HDRLEN	3	OPTION
MAXOPT	15	OPTION
MAXPGM	106	OPTION
MENUMX	20	OPTION
OFNLEN	10	OPTION
OPTLE	9	OPTION
PGMLEN	10	OPTION
MESVEC	1	PVFILE
CLPBMS	5	RNKORD
CLSBMS	4	RNKORD
DC	1	RNKORD
DSPCHR	8	RNKORD
MBCLPR	3	RNKORD
MBYCLS	2	RNKORD
NDIMEN	3	RNKORD
NUMCLS	4	RNKORD
OPTNUM	2	RNKORD
OVRALL	1	RNKORD
RNKOPT	5	RNKORD
RNKPRM	6	RNKORD
SRTFLG	7	RNKORD

OLPARS Parameter Limits -- H
According to File Location

Parameter	Value	Location
A	7	SCREEN
B	8	SCREEN
C	9	SCREEN
D	10	SCREEN
DS	19	SCREEN
E	15	SCREEN
F	16	SCREEN
G	17	SCREEN
HC	6	SCREEN
HD	4	SCREEN
HSN	2	SCREEN
LCH	13	SCREEN
MRW	11	SCREEN
NC	18	SCREEN
NCL	12	SCREEN
NOSCRN	19	SCREEN
SC2NUM	13	SCREEN
WC	5	SCREEN
WD	3	SCREEN
WS	1	SCREEN
MAXMTR	10	SMFILE
APPEND	2	STANDARD
CMDLEN	10	STANDARD
DATA	1	STANDARD
DELETE	1	STANDARD
EOF	-1	STANDARD
EOS	0	STANDARD
ERROR	-1	STANDARD
FNMLEN	34	STANDARD
HEADER	0	STANDARD
HELP	-2	STANDARD
LOGIC	2	STANDARD
MAXLIN	133	STANDARD
NEW	1	STANDARD
NFIXFL	10	STANDARD
NODLEN	4	STANDARD
OLD	0	STANDARD
QUIT	-1	STANDARD
READ	0	STANDARD
SAVE	0	STANDARD
TRELEN	8	STANDARD
WRITE	1	STANDARD
BPET	12	TIFILE
CAPSIZ	12	TIFILE
ENMEAN	17	TIFILE
FCP	7	TIFILE
FLNP	8	TIFILE
FRELST	13	TIFILE
LNBP	10	TIFILE
LNLP	9	TIFILE
NL	4	TIFILE
NLNODE	2	TIFILE
NOKIDS	3	TIFILE
NOVECS	1	TIFILE
PP	5	TIFILE
SP	6	TIFILE
TABSIZ	100	TIFILE
TIHSIZ	105	TIFILE
VP	11	TIFILE
TVHENE	11	TVFILE

APPENDIX J

File type naming conventions of CLPARS files

This text is dedicated to the cause of 'keeping straight' all the different file name types (or file extension names) found within the RSX-11M version of portable CLPARS. As is known by those people who are familiar with DEC's RSX, a file specification looks like:

<filename>.<type>;<version number>

where <filename> is a 0-9 character string of letters and/or digits,

<type> is a 0-3 character string of letters and/or digits, and

<version number> is an octal number, used in uniquely identifying files with the same <filename> and <type>.

This text will only deal with the <type> portion of the RSX filename.

First, you must be introduced to the abbreviations used here. The following mneumonics refer to specific software tools used in constructing CLPARS.

AFT - PAR's FORTRAN preprocessor

DSR - DEC Standard Runoff (VMS text formatter)

F4P - DEC's FORTRAN IV Plus compiler

INC - PAR's 'include' utility program

LBR - DEC's object module librarian

MAC - DEC's macro assembler

MUN - another name for TEC

PIP - DEC's file transferring program

RNO - (runoff) a text formatter

TEC - TECo (text editor and corrector)

TKB - DEC's task builder (linker)

We can now proceed to the file type (extension) naming conventions used in the portable OLPARS project. Note that the period (.) portion of the file specification will preceed all the following <type> names (this is to remind you that the 3 characters represent file extension names).

.AFT - OLPARS programs to be 'AFT' processed

.ALG - textual algorithms to be processed by 'RNO'

.ASM - OLPARS assembly files that must be 'INClude' processed

.BLD - task build (linker) command files

.CMD - RSX-11M '@' processor command files

.DCL - AFT or assembly declaration files (to be 'INCluded')

.DOC - text files already processed by the 'RNO' utility

.FIG - OLPARS figure documents (post 'RNO' processed, pre 'INC')

.FTN - FORTRAN files to be processed by 'F4P'

File type (extension) naming conventions -- J

.HDR - documentation headers for OLPARS programs (to be 'INCluded')

.INC - 'INClude' processor input files

.MAN - manuals after 'RNO' processing

.MAC - assembly files to be processed by 'MAC'

.OBJ - program object modules used by task builder and librarian

.ODL - task build (linker) overlay description files

.OLP - OLPARS binary information files

.PAM - OLPARS programmer aide documents before 'RNO' processing

.PIC - pictures (figures) to be processed by 'RNO'

.PRM - OLPARS programmer's reference manual before 'RNO' processing

.PTH - path name files used by the 'INClude' processor

.RNO - text files to be processed by the 'RNO' utility

.SPC - OLPARS program specification documents (to be 'RNO' processed)

.SRM - OLPARS software reference manual before 'RNO' processing

.SRC - OLPARS 'AFT' source files that must be 'INClude' processed

.TEC - TECo command files

.TPL - template files used in generating documentation or command files

.TSK - task images of executable programs

.TXT - Miscellaneous text files

.USM - OLPARS user manual documents before 'RNO' processing

About Files and Processes

The chart on the following page shows the relationships between the previously mentioned CLPARS file types and processes. In the chart, processes are represented with boxes, file types are denoted with an initial period (.). The @ symbol surrounding processes and file types indicates which operations are performed by the RSX-11M command file processor. Lines intersecting process boxes show that the file at the other end of the line is used as input to the process ("Arrowed" lines are used, also).

```
                          .CMD
                            |
                       ----|----
                      | AT. (@) |
                       ---------
                           | |
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                                      @
@              ---------                 ---------                     @
@  -- .HDR <-- |   MUN   | <- .DOC  <-- |   RNO   | <- .SPC           @
@  |           ----|----                 ---------       .ALG          @
@  |               |                                                   @
@  |-- .DCL    COMMENT.TEC                                             @
@  |               |                                                   @
@  |           ----|----                 ---------                     @
@  |-- .FIG <-- |   MUN   | <- .DOC  <-- |   RNO   | <- .PIC           @
@  |           ---------                 ---------                     @
@  |                                                                   @
@  |      .SRC                                                         @
@  |        |                                                          @
@  |    ----|----          ---------             ---------             @
@  ->|     INC    | --> .AFT -> |   AFT    | --> .FTN -> |   F4P    |  @
@       ----|----          ---------             ---------             @
@           |                                        |                 @
@         .PTH                                       |                 @
@           |                                        |                 @
@       ----|----          ---------                 |                 @
@    |    INC    | --> .MAC -> |   MAC    | -->    .OBJ   <---         @
@       ----|----          ---------              /  \                 @
@           |                                    /    \                @
@         .ASM             (main programs) /        \ (subrtns.)       @
@                                          /          \                @
@                                   -------/-         -\-------        @
@                         .TSK <-- |   TKB   |       |   LBR   |       @
@  Relationships between            ----|----         ---------        @
@   files and processes                 |                 |            @
@                                     .BLD  .CLB <---------            @
@                                     .ODL                             @
@                                                                      @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                                      @
@    ---------          ---------           ---------                  @
@  |   INC    |--> .PRM ->|   PIP    |--> .RNO ->|   DSR    |--> .MAN  @
@    ---------          -|-------           ---------                  @
@                        |- .USM                                       @
@                        |- .PAM                                       @
@                         - .SRM                                       @
@                                                                      @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

                     ---------
   Miscellaneous <-- |  .TSK   | <-- .CLP
   text files        ---------        .TXT (some, not all)
```

APPENDIX K

Miscellaneous Text Files Created by OLPARS Commands

The following list contains the names of the various text files created by OLPARS commands along with the name of the creating commmand (in parenthesis). Note, these files have no <type> name.

CLASIFY1	-	list of logic nodes and number of vectors assigned to each node. (LOGEVAL)
CLASIFY2	-	list of vector identifiers and logic nodes to which they were assigned. (LOGEVAL)
CONMAT	-	confusion matrix listing. (LOGEVAL, NMEVAL, PWEVAL)
EIGEN	-	eigenvalue listing. (L1EIGV, L2EIGV, REPROJECT, S1EIGV, S2EIGV)
LOGDUMP	-	logic tree dump. (PRTLOG)
MISCLASS	-	vector misclassification listing. (CREATLOG, LOGEVAL, NMEVAL, PWEVAL)
SAVMTR	-	a saved matrix listing. (MATRIX)
TREEDUMP	-	data tree dump. (PRTDS)
VECIDS	-	new-old vector identifier relationship listing. (MAKETREE)

INDEX